Saimaan ammattikorkeakoulu
Tekniikka, Lappeenranta
Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautumisvaihtoehto

Evgeni Kovalev

INNOMOBIILI-TIEDONKERUUJÄRJESTELMÄ

Opinnäytetyö 2011

TIIVISTELMÄ
Evgeni Kovalev
Innomobiili-tiedonkeruujärjestelmä, 60 sivua, 1 liite
Saimaan ammattikorkeakoulu, Lappeenranta
Tekniikka, Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautumisvaihtoehto
Opinnäytetyö 2011
Ohjaajat: Lehtori Martti Ylä-Jussila, Saimaan ammattikorkeakoulu.
Toimitusjohtaja Markus Heikkinen, Innotek Oy

Opinnäytetyön aiheena on suunnitella ja toteuttaa Innomobiili-tiedonkeruujärjestelmää Innotek Oy:n työntekijöiden käyttöön. Projektin asiakas on liekasalainen LVI-alan yritys, jonka toiminta-ajatus on kehittää ja markkinoida vedenkäyttötehokkuutta parantavia teknisiä ratkaisuja. Organisaatio käyttää liiketoiminnassaan Energo-ohjelmaa, joka käsittää vedenkulutusta pienentäviä LVI-palveluja ja ratkaisuja.

Energo-ohjelmaan liittyvä tietojenkäsittely on hoidettu toistaiseksi pelkästään normaaleilla toimisto-ohjelmilla, mikä on työlästä ja toisaalta työnohjaus ei ole riittävän reaaliaikaista. Yrityksen johto on aloittanut vuonna 2004 suunnitella sähköistä raportointimenetelmää, jonka tarkoituksena on parantaa työaikaseurantaa, materiaalin käyttöseurantaa sekä kiinteistöhallintaa.

Työssä on suunniteltu ja toteutettu Innomobiili-tiedonkeruujärjestelmä Jaakko Purhosen laatiman toiminnallisen määrittelyn pohjalta.

Kehitystyössä on käytetty MS Visual Studio 2008 -kehitysympäristöä, StarUML-mallinnustyökalua, SVN-versiohallintajärjestelmää, C#-ohjelmointikieltä ja testausalustana HTC Touch HD2 -älypuhelinta.

Työn päätteeksi on saatu merkittäviä tuloksia Innomobiilin käyttöliittymään, toteutettu lomakkeita ja osittain niiden takana olevaa logiikkaa. Lisäksi on saatu aikaan toimiva XML-RPC-tiedonsiirto, mediavälitys sekä GPS-seuranta. Myös palvelinpuolen komponentista on tehty toimiva versio. Järjestelmän määrittelydokumenttia on päivitetty muun muassa käyttötapausten ja ulkoisten liittymien suhteen. Järjestelmästä on laadittu suunnitteludokumentti, jossa kuvattu toteutuksen kannalta oleellisimmat asiat.

Avainsanat: Innotek Oy, Energo, Innomobiili, Windows Mobile.

ABSTRACT
Evgeni Kovalev
Innomobiili data collection system, 60 pages, 1 appendix
Saimaa University of Applied Sciences, Lappeenranta
Technology, Information Technology
Software Engineering
Bachelor's Thesis 2011
Instructors: CEO Markus Heikkinen, Innotek Corp.
Lecturer Martti Ylä-Jussila, Saimaa University of Applied Sciences.

The goal of this bachelor's degree thesis was to design and implement Innomobiili-data collection system, which will be used by the Innotek Ltd's employees. The customer of this project is Innotek Ltd, the HVAC company from Lieksa. The business idea of the company is to develop and market efficiency-enhancing technical solutions for water use. The organization is using in its marketing the Energo named efficiency program which consists of HVAC products and services that help to save water.

Reports were made so far with the basic tools, making the use of resources opaque, catching more attention to unnecessary adjustments. In 2004, the company's management has begun to plan an electronic reporting system, which aims at improving the working time tracking, material usage monitoring, as well as estate management. This project is targeting on developing the Innomobiili data collection system using predefined specification document made by Jaakko Purhonen.

To develop Innomobiili such tools as MS Visual Studio 2008 - IDE, StarUML-modeling tool, and SVN-version control system were used. The C # was used as a programming language and HTC Touch HD2 smart phone was used as deploying and testing platform.

In the end of the work the significant results were achieved in developing the interface and the logics of Innomobiili. The working version of server-side components was also developed during the project. In addition, the specification document of the project was expanded with significant aspects and additions. The software design document was also produced during the project.

Keywords: Innotek Ltd, Energo, Innomobiili, Windows Mobile.

TERMIT JA LYHENTEET

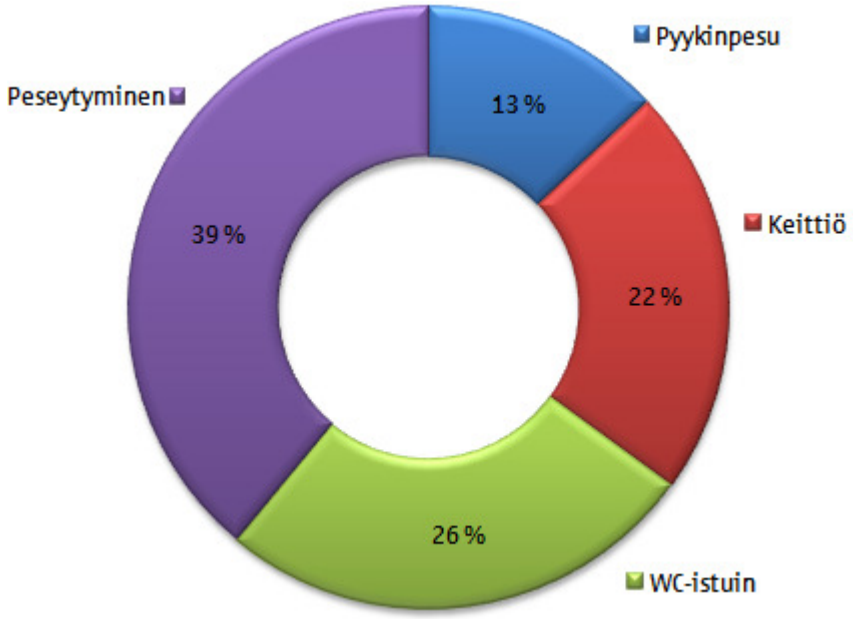
Elementti	Tässä työssä elementillä tarkoitetaan UML-kaavion mitä tahansa jäsentä, paitsi assosiaatiosuhteita. Näin olleen elementillä voidaan tarkoittaa UML-kaaviossa olevaa luokkaa, oliota, komponenttia, pakkausta jne.
IEEE	Institute of Electrical and Electronics Engineers on maailmanlaajuinen tekniikan alan järjestö, jonka toimintaan kuuluu laaja julkaisutoiminta, tieteellisten konferenssien järjestäminen, koulutuksen edistäminen sekä monien alan keskeisten standardien määrittely.
MSIL	Microsoft Intermediate Language on Microsoft Oy:n kehittämä alustariippumaton kieli, jonka .NET-kääntäjä tuottaa mistä tahansa .NET-yhteensopivasta ohjelmakoodista.
MySQL	Suosittu ja tehokas SQL-tietokannan hallintajärjestelmä.
Objekti	Tässä työssä objektilla tarkoitetaan luokan oliota. Sana johtaa juurensa englanninkielen sanasta *object*.
OMG	Object Management Group on kansainvälinen työryhmä, joka kehittää ja ylläpitää alustariippumattomia oliopohjaisia suunnitteluratkaisuja organisaatiotasolla. OMG ei tavoittele taloudellista etua toiminnallaan.
Relaatio	Tässä työssä relaatiolla tarkoitetaan UML-kaavion elementtien välisiä suhteita.
Singleton	Suunnittelumalli, jossa luokasta voi olla vain yksi instanssi. Hyödyllinen ratkaisu esimerkiksi, kun verkkojonossa jaetaan yhtä resurssia asiakasta kohti.
SOLID	Single responsibility, Open-closed, Liskov substitution, Interface segregation, Dependency inversion on Robert Martin C:n vuonna 2000 esittämät periaatteet helposti ylläpidettävien ja laajennettavien järjestelmien suunnitteluun ja kehitykseen.
StarUML	Ilmainen UML-mallinnusohjelma Windows-alustalle.

SQL	IBM:n kehittämä standardoitu käskykieli, jolla relaatiotietokantaan voi tehdä erilaisia hakuja, muutoksia ja lisäyksiä.
Subversion	Avoimen lähdekoodin versiohallintajärjestelmä, joka pitää kirjaa tiedostoihin hajautetusti tehdyistä muutoksista. Subversion (lyh. *SVN*) korvaa vanhentuneen CVS-versiohallintajärjestelmän.
TortoiseSVN	Subversion-versiohallintajärjestelmän ilmainen asiakasohjelma, joka toimii Windows-alustalla. TortoiseSVN on integroitu Windows:n kontekstivalikkoon ja tarvittaessa on saatavilla integrointiliitännäiset MS Visual Studioon.
UML	Unified Modeling Language on OMG:n vuonna 1997 standardoima graafinen mallinnuskieli, joka sisältää joukon graafisia notaatioita, joiden avulla luodaan visuaalisia malleja oliopohjaisille järjestelmille. Uusin standardi UML 2.3 ilmestyi toukokuussa 2010.
XML-RPC	Remote Procedure Call, on UserLand Software ja Microsoftin kehittämä tietoliikenneprotokolla, joka mahdollistaa etämetodien kutsun. Siirrettävä tieto on XML-muodossa ja kulkee verkossa HTTP-protokollan avulla.
Älypuhelin	Matkapuhelin, jossa on perinteisten mobiilipuhelintoimintojen lisäksi kämmentietokonetta muistuttavia ominaisuuksia.

SISÄLTÖ

1 JOHDANTO

Jokainen suomalainen kuluttaa vuorokaudessa keskimäärin 155 litraa vettä (Kuva 1.1). Tästä määrästä yksi asukas käyttää tiskatessaan 35 litraa ja pyykkiä pestessään 20 litraa vettä. Vessasta vedetään alas joka päivä melkein 40 litraa puhdasta juomavettä. Eniten vettä kuuluu kuitenkin peseytymisessä - noin 60 litraa vuorokaudessa.



Kuva 1.1 Yhden asukkaan vuorokautinen vedenkulutus. (Motiva 2011)

Veden puhdistamiseen, lämmittämiseen ja siirtämiseen kuuluu energiaa, jonka tuotannosta syntyy ilmastonmuutokseen vaikuttavia hiilidioksidipäästöjä. Tällöin säästämällä vettä voidaan pienentää siihen kuluvan energian määrää, jolloin saadaan puhtaampi ympäristö. Vesivuotojen korjauttamisella ja vettä säästävien laitteiden käytöllä varmistetaan, että kaikki ostetusta vedestä hyödynnetään täysimääräisesti. Jokainen asukas voi säästää vettä pienelläkin muutoksella vedenkäyttöönsä.

Lieksalainen LVI-alan yhtiö Innotek Oy on tuonut markkinoille Energo™- säästöohjelman, joka sisältää vettä säästäviä LVI-ratkaisuja ja palveluja.

Lyhykäisyydessään Energo-ohjelma käsittää kiinteistön vesikalusteiden kartoittamisen ja virtausmittauksen tekemisen, joiden perusteella suoritetaan vettä säästäviä LVI-asennuksia sekä laaditaan kiinteistön kuntoraportti.

Kuntoraportti on eräänlainen todistus kiinteistön vesikalusteiden kunnosta. Sen perusteella voidaan tarvittaessa tilata lisäasennuksia.

Asentajaraportit on tehty tähän mennessä kynällä ja raporttilomakkeilla, mikä aiheutti hidasteita yrityksen liiketoiminnassa. Vuonna 2004 yrityksen johto on aloittanut suunnittelemaan sähköistä raportointijärjestelmää, jonka tarkoituksena on parantaa työaikaseurantaa, materiaalin käyttöseurantaa sekä kiinteistöhallintaa. Järjestelmän tiedonkeruuosio tulee toimimaan paperilomakkeiden sijasta työntekijän mukana kätevästi kulkevana mobiililaitteen sovelluksena.

Monimutkaisen kokonaisuuden hallitsemiseksi Innotek Oy:n tavoitteena on kehittää toiminnanohjausjärjestelmä, jonka tarkoituksena on toimia hajautetusti niin kentällä kuin toimistossa.

Toiminnanohjausjärjestelmä koostuu XAMPP-verkkopalvelimella toimivasta Innonet- sekä Windows Mobile -alustalla toimivasta Innomobiilitiedonkeruujärjestelmästä.

Opinnäytetyön tavoitteena on suunnitella ja toteuttaa Innomobiilitiedonkeruujärjestelmä pohjautuen Jaakko Purhosen opinnäytetyönään laatimaan toiminnalliseen määrittelyyn.

2 INNOTEK OY

Innotek Oy on vuonna 1995 Lieksaan perustettu LVI-alan yhtiö, joka tähtää veden ja energian kulutuksen säästöön tarjoamalla asiakkaalleen innovatiivisia LVI-tuotteita ja palveluita. Yhtiöllä on yhteistyökumppaneita eri puolella Suomea ja sen palvelut on suunnattu sekä yksityis- että yritysiakkaille. Kymmenen henkilön yritystiimin toimitusjohtajana sekä tämän opinnäytetyöprojektin asiakkaana toimii Markus Heikkinen.

2.1 Energo™-säästöohjelma

Innotek Oy:n liiketoiminta perustuu Energo-toimintamalliin, jonka tarkoituksena on pienentää vedenkulutusta yrityksen kehittämien LVI-tuotteiden ja palveluiden avulla. Ohjelma käsittää kahdeksan vaihetta seuraavassa järjestyksessä:

1. Kiinteistön edustaja tekee Energo-tilauksen.
2. Energo-asentajat suorittavat kiinteistössä
   - kalustokartoituksen
   - virtausmittaukset
   - huoltosuositukset.
3. Energo-tilauksen laajuus tarkennetaan kartoituksen tulosten perusteella.
4. Sovitaan asennusaika ja tiedotetaan kiinteistöyhtiössä.
5. Energo-asentajat suorittavat asennustyön ja kuntoraportoinnin.
6. Asennus- ja kuntoraportit toimitetaan laskun mukana isännöitsijälle.
7. Isännöitsijä tarkastaa kuntoraportit ja tilaa tarvittaessa lisäkorjaustyöt.
8. Suoritetaan mahdolliset lisäkorjaustyöt.

(Innotek Oy 2000).

2.2 Energo-kartoitus

Energo-kartoituksen tavoitteena on muodostaa luettelo asennettavista varaosista ja määrittää palveluja, joiden avulla saavutetaan haluttu veden säästötaso. Yhtiön koulutetut Energo-asentajat kartoittavat kiinteistössä olevaa LVI-kalustoa ja tekevät hanojen ja suihkujen virtausmittauksia. Tarvittaessa asentajat mittaavat vuorokauden kokonaiskulutuksen ja tarkastavat kiinteistön päävesimittarin. Energo-kartoituksen tuloksena syntyy kartoitusraportti, jonka pohjalta asiakkaalle tiedotetaan tarvittavista asennustoimenpiteistä asennustarjouksen muodossa.

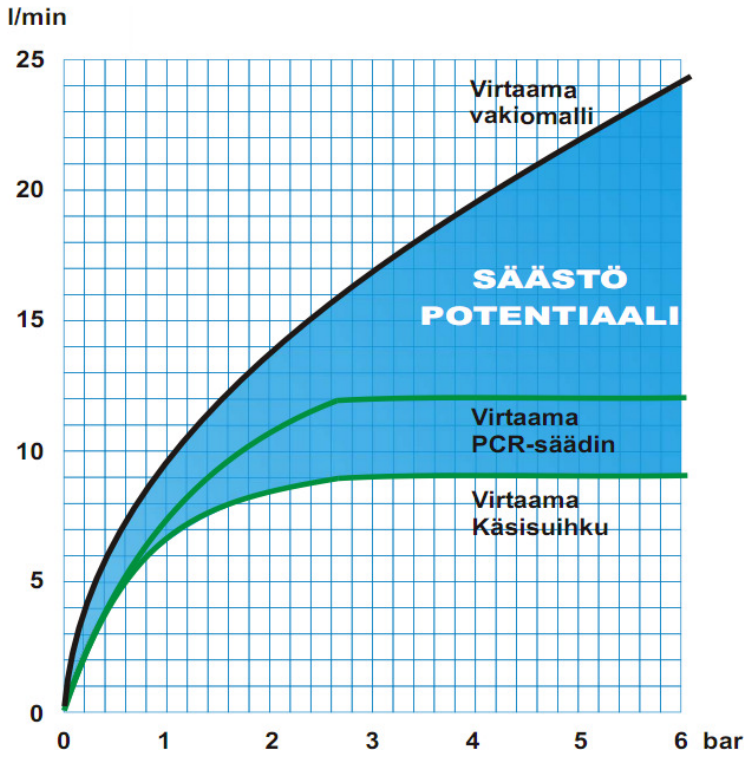
2.3 Asennustyö

Energo-säästöohjelman keskeinen vaihe on asennustyö, johon päädytään asiakkaan hyväksyttyä asennustarjouksen. Asennus käsittää kartoitusraportissa mainittujen osien paikalle asentamisen ja korjaustoimenpiteiden suorittamisen. Asennuksen aikana asentaja pitää kuntoraporttia, johon kirjataan vesikalusteiden vuodot, vuotojäljet rakenteissa sekä asukkaiden ilmoittamat puutteet ja viat. *Kuntoraportti™* on Innotek Oy:n tuote, joka on isännöitsijälle tai omistajalle ilmainen ja se toimii kohteen kuntotodistuksena. Lisäksi kuntoraportin perusteella voidaan helposti tilata tarvittavia korjaustöitä.

2.4 Huoltopalvelut

Innotek Oy tarjoaa asiakkailleen lukuisia asennuspalveluja, joita voidaan tilata kuntoraportin pohjalta. Ratkaisu WC-istuinten huomaamattomiin vesivuotoihin on *VuotoPois™*-huoltopalvelu. Asentaja uusii ja puhdistaa WC-istuimen vesitankin tärkeimmät tiivisteet ja säätää huuhteluvesimäärää asentamalla *Vetovahti™*-säästölaitteen. *Vetovahdin* avulla huuhteluveden määrää pienenee puoliksi korkeamman huuhtelupaineen ansiosta.

Suihkuveden virtaamaa vakioimaan yritys toimittaa asiakkailleen *Käsisuihku™* ja *PCR™*-vakiovirtausventtiilejä, jotka tasaavat vedenvirtaaman säilyttäen käyttömukavuuden. *Käsisuihkun* rungossa sijaitseva vakiovirtausventtiili tasaa virtaaman 9 litraan minuutissa ja *PCR*-säädin puolestaan tasaa virtaaman 12 litraan minuutissa. Kuva 2.1 esittää säästöpotentiaalin, joka on parhaiten näkyvissä, kun seurataan vakiovirtausventtiilien virtaamia vedenpaineen suhteen, verrattuna tavanomaisiin LVI-laitteisiin.



Kuva 2.1 Vakiovirtausventtiilien tuottama säästöpotentiaali. (Innotek 2000)

Hanojen 7,5 litran vedensäästö tavanomaisen 15 litran sijasta saavutetaan asentamalla asiakaskohteisiin *Säästömalli E™*- ja *Säästömalli AC™*-poresuuttimia. Jälkimmäinen malli toimitetaan kohteisiin, joissa kalkin muodostus on voimakasta. Säästöporesuuttimet säästävät energiaa ja vettä vähentämättä vedenkäytön mukavuutta. Kuva 2.2 esittää *Säästömalli E* ja *Säästömalli AC* poresuuttimien rakenteiden läpileikkauksia.

Kuva 2.2 Poresuuttimen *Säästömalli E* ja *Säästömalli AC (edessä)* rakenteiden läpileikkaukset. (Innotek 2000)

3 OHJELMISTOTUOTANNON TEKNIIKAT JA MENETELMÄT

Ohjelmistotuotanto tarkoittaa kurinalaista prosessien kokonaisuutta, jossa sovelletaan toimiviksi osoitettuja tekniikoita ohjelmistoratkaisujen tuottamiseen suunniteltujen aika-, laatu- ja kustannusvaatimuksien puitteissa.

3.1 Tekniikoista

Ohjelmistotuotanto käsittää kurinalaisia prosesseja, joihin sovelletaan erilaisia toimiviksi osoitettuja tekniikoita laadukkaiden ja helposti ylläpidettävien ohjelmistotuotteiden aikaansaamiseen. Oikean tekniikan käyttö oikeassa kehitysvaiheessa auttaa valmistamaan tuotetta ajoissa ja sen valmistukseen rajatuilla resursseilla. Vain harvat toimiviksi osoitetut tekniikat muuttuvat lopulta standardeiksi, silloin dokumentaatio selkenee ja kehittäjävälinen kommunikaatio paranee tuotantovaiheiden välissä.

Voidaan siis ajatella, että standardinmukaisten tekniikoiden käyttö tarkoittaa joksikin määrin mutkattoman, läpinäkyvän ja muunneltavan ohjelmiston tuottamista.

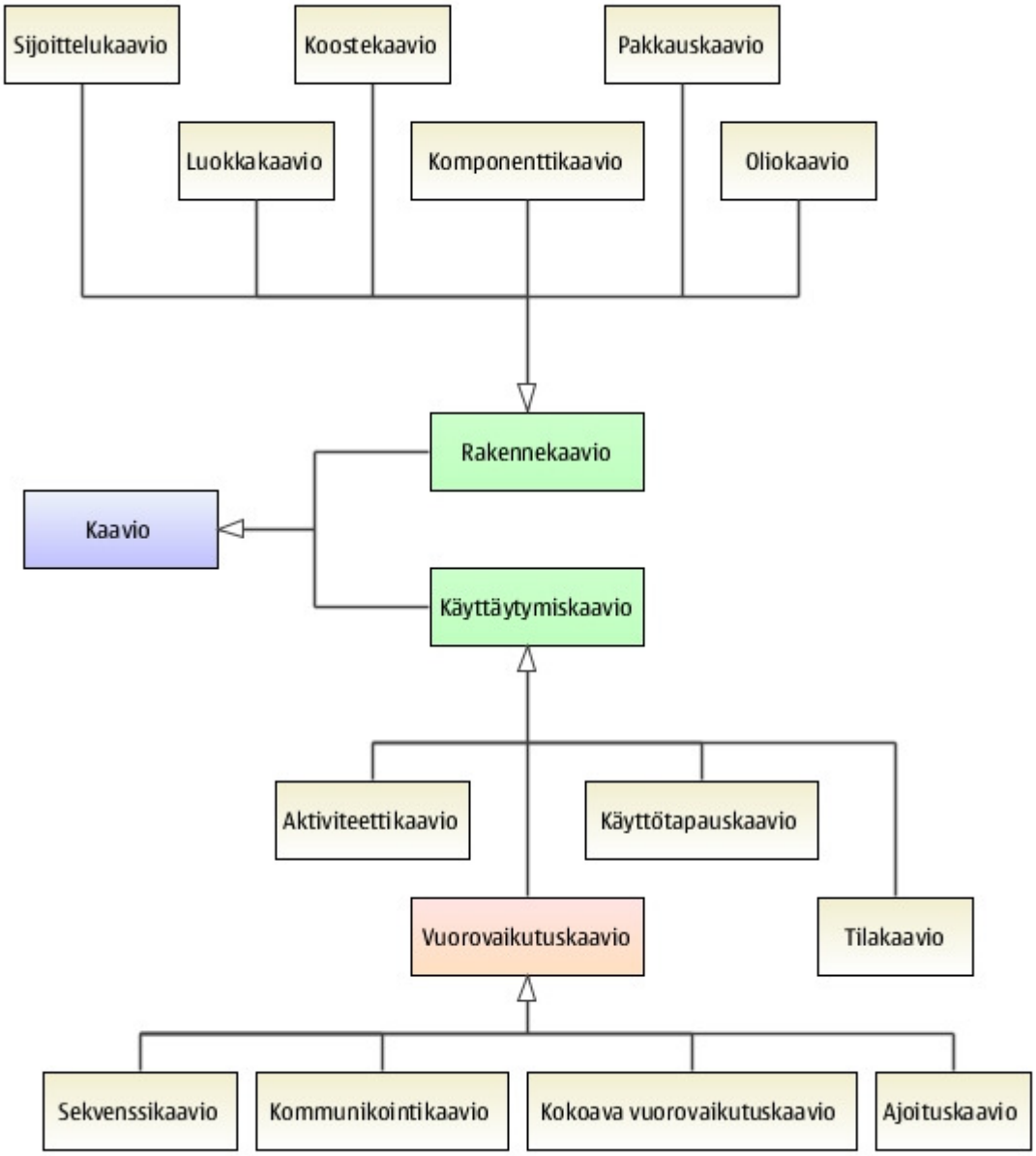
3.2 Mallinnustekniikat ja UML

Ohjelmistotuotannon eri prosessit kommunikoivat keskenään spesifikaatioiden, kaavioiden ja kuvioiden avulla. Yksinkertaisia vaatimuksia voidaan toki esittää pelkällä tekstillä, mutta monimutkaisemmat rakenteet vaativat kuvitettua esitystä. Asioiden ymmärtämisen ja muistamisen lisäksi kuviot herättävät muistista aiemmin opittua malleja, joita voidaan soveltaa käsiteltävän ongelman ratkaisuun.

Asioita voidaan kuvata monella eri tavalla, ideaalitapauksessa kehittäjän pitäisi ymmärtää, mitä asiakas tarkoittaa, ja pystyä välittämään tietoa eteenpäin ilman poikkeamia alkuperäisistä vaatimuksista. Ongelman ratkaisee standardoitu, moniulotteinen kuvaustekniikka, joka poistaa kehittäjienvälisen kielimuurin ja kuvaa ongelmaa sekä sen ratkaisua riittävän tarkasti eri näkökulmista.

UML-mallinnuskieli, jonka alun perin standardoi OMG vuonna 1997, on osoittautunut toimivaksi ratkaisuksi monimutkaistenkin järjestelmien mallintamiseen. Uudempi standardi UML 2.3 ilmestyi toukokuussa 2010. UML-kielen avulla voidaan kuvata monipuolisesti järjestelmän rakennetta, käyttäytymistä ja vuorovaikutusta ulkopuolisten järjestelmien kanssa. UML-mallinnuskieli on riittävän tarkka, jotta sitä voi muuttaa automaattisesti (eng. *forward engneering*) suoraan ohjelmakoodiksi, mikä antaa mahdollisuuden tuottaa järjestelmän koodirunkoa jo varhaisessa arkkitehtuurisuunnittelun vaiheessa.

UML 2.3 -standardiin kuuluu 13 kaaviotyyppiä, jotka ovat järjestäytyneet kuvan 3.1 hierarkian mukaisesti.



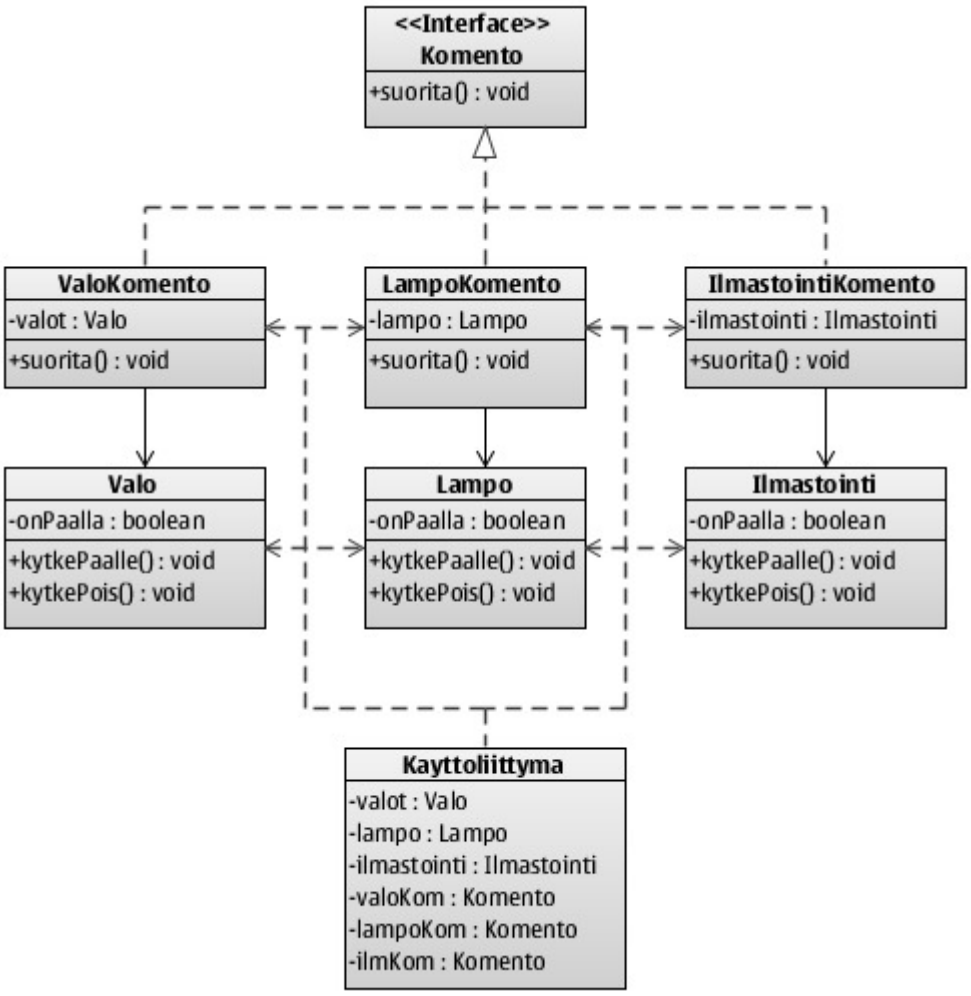
Kuva 3.1 UML-kaaviohierarkia. (Agilemodeling.com 2009)

UML-kaaviot mahdollistavat järjestelmän ja sen osien tarkastelun eri näkökulmista. Ne jakaantuvat kolmeen ryhmään: rakenne-, tila- ja käyttäytymistyyppisiin kaavioihin. Rakennetyppiset kaaviot havainnollistavat järjestelmän kiinteitä rakenneosia ja käyttäytymistyyppiset kaaviot esittävät järjestelmän sisäistä ja ulkoista vuorovaikutusta: mikä aiheuttaa tietyn toiminnon ja mitkä seuraamukset toiminto aiheuttaa.

Käyttäytymisryhmän vuorovaikutuskaaviot muodostavat oman aliryhmänsä, jonka kaaviot kuvailevat järjestelmään kohdistuvaa ulkopuolista ärsytystä, esimerkiksi käyttäjän syötteitä. Seuraavaksi käsitellään konkreettisemmin UML-kaavioita ja niiden elementtejä.

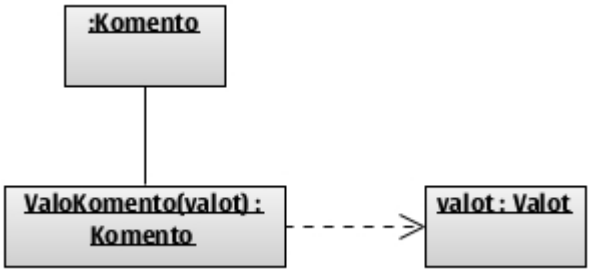
3.2.1 Rakennekaaviot

Luokkakaavio esittää järjestelmän luokkarakennetta ja luokkien välisiä suhteita. Kaavioon voivat osallistua luokat, attribuutit, metodit, rajapinnat ja relaatiot. Kuva 3.2 esittää esimerkin luokkakaaviosta.



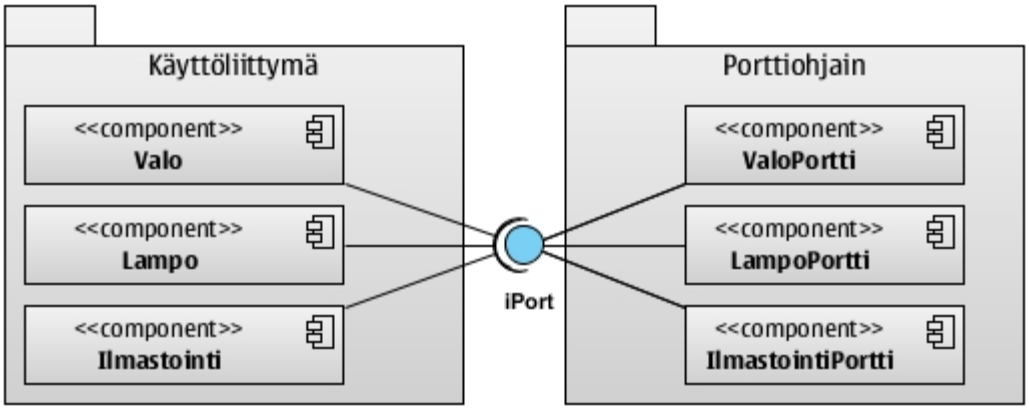
Kuva 3.2 Luokkakaavio.

Oliokaavio esittää järjestelmän ajonaikana syntyviä olioita ja niiden kommunikointia. Kaavion avulla seurataan olioiden kehitystä ja kommunikointia tietyltä ajanjaksolta. Oliokaavion avulla voidaan simuloida ja testata haluttuja erikoistilanteita. Kaavioon voivat osallistua oliot, niiden attribuutit, funktiot ja relaatiot. Kuva 3.3 esittää esimerkin oliokaaviosta.



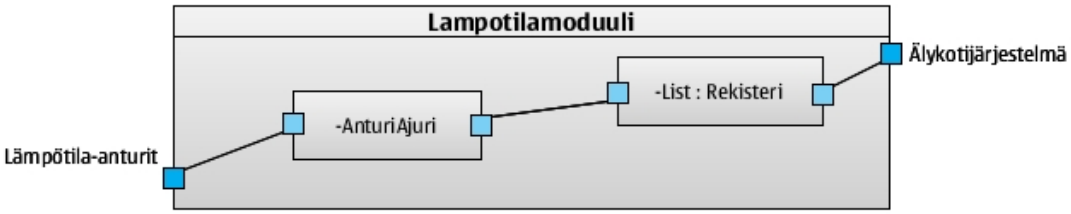
Kuva 3.3 Oliokaavio.

Komponenttikaavio kuvaa järjestelmän komponentteja ja niiden suhteita. Kaavioon voivat osallistua komponentit, rajapinnat, tiedostot, kirjastot, moduulit tai pakkaukset. 3.4 esittää esimerkin komponenttikaaviosta.



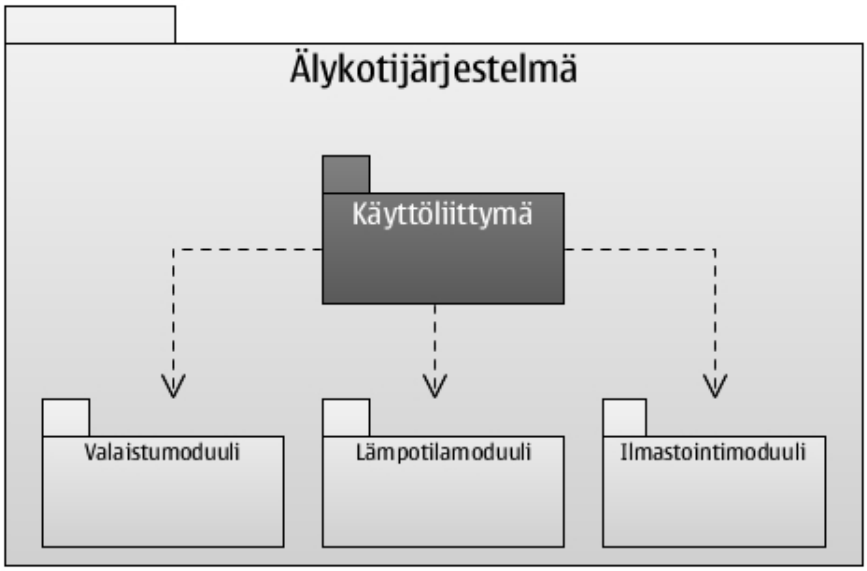
Kuva 3.4 Komponenttikaavio.

Koostekaavio esittää luokkien sisäistä rakennetta ja toiminnallisuutta. Kaavio on hyvä tapa esittää suunnittelumallien toimintaa. Kaavioon voivat osallistua luokat, attribuutit, funktiot, rajapinnat tai relaatiot. Kuva 3.5 esittää esimerkin koostekaaviosta.



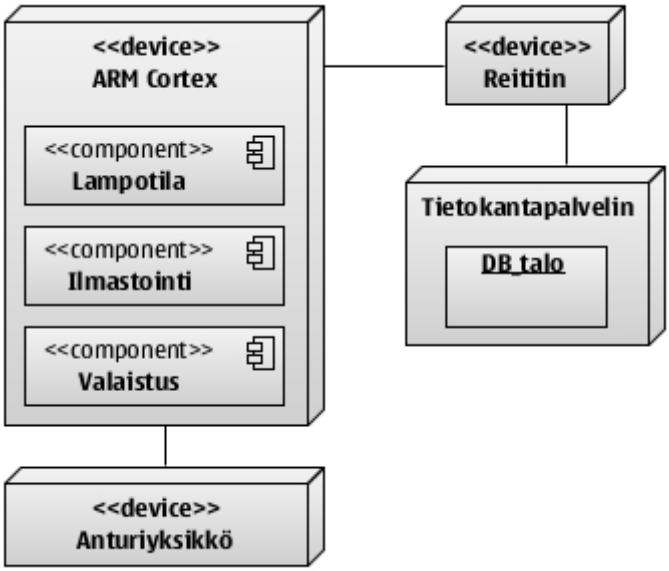
Kuva 3.5 Koostekaavio.

Pakkauskaaviot esittävät järjestelmän arkkitehtuurista kerrostumista. Pakkausten välisillä suhteilla voidaan kuvata siirtymismekanismeja arkkitehtuurikerrosten välillä. Kaavioon voivat osallistua pakkaukset, toiset kaaviot sekä pakkausten väliset suhteet. Kuva 3.6 esittää esimerkin pakkauskaaviosta.



Kuva 3.6 Pakkauskaavio.

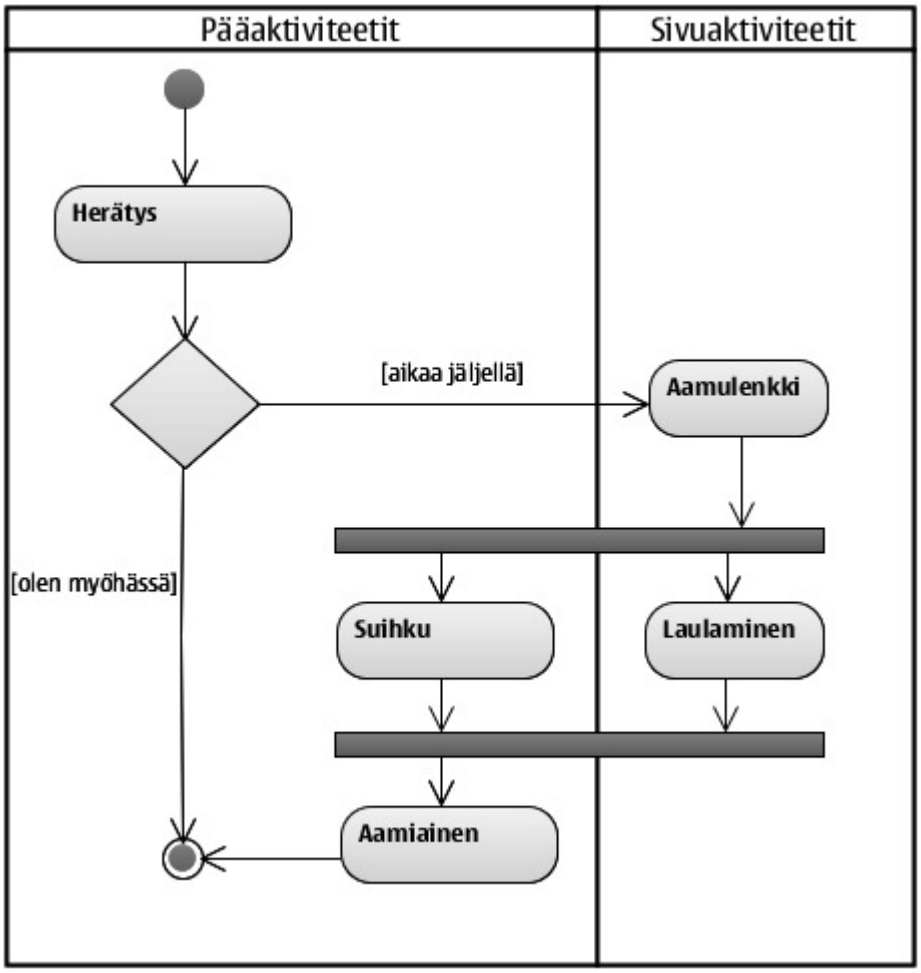
Sijoittelukaavio havainnollistaa järjestelmää fyysisellä tasolla, tosin sanoen mitkä fyysiset solmut (eng. *node*) ja millä tavalla osallistuvat järjestelmän toimintaan. Kaaviossa voivat esiintyä toimijat, palvelimet, tiedostot, pakkaukset, tietokannat, verkot ja muut elementit. Kuva 3.7 esittää esimerkin sijoittelukaaviosta.



Kuva 3.7 Sijoittelukaavio.

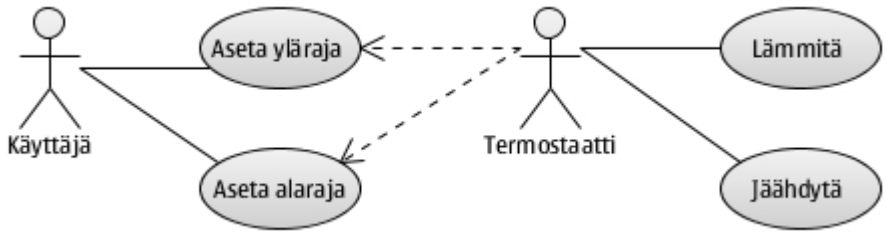
3.2.2 Käyttäytymiskaaviot

Aktiviteettikaavio kuvaa järjestelmän tapahtumien yksityiskohtaista etenemistä. Kaavioon osallistuu yksi toimija, järjestelmän komponentit sekä tapahtuman alku- ja loppupiste. Kuva 3.8 esittää esimerkin aktiviteettikaaviosta.



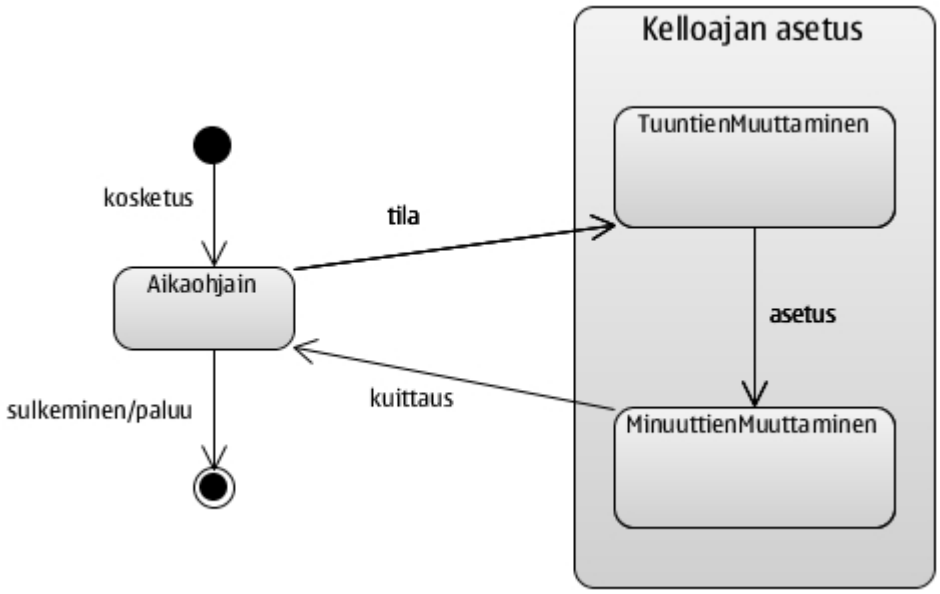
Kuva 3.8 Aktiviteettikaavio.

Käyttötapauskaavio kuvaa järjestelmän ulkopuoliset toimijat ja niiden suhteet järjestelmän käyttötapauksiin. Kaavion tarkoituksena on antaa asiakkaalle, kehittäjälle ja loppukäyttäjälle perusteellisen kuvan järjestelmän käyttäytymisestä ja toiminnoista. Laadukas käyttötapaus alkaa käyttäjäroolin aloitteesta ja päättyy järjestelmän tuottamaan lisäarvon käyttäjälleen. (Haikala ym. 2006) Kaavion osallistuvat käyttötapaukset, toimijat sekä relaatiot. Kuva 3.9 esittää esimerkin käyttötapauskaaviosta.



Kuva 3.9 Käyttötapauskaavio.

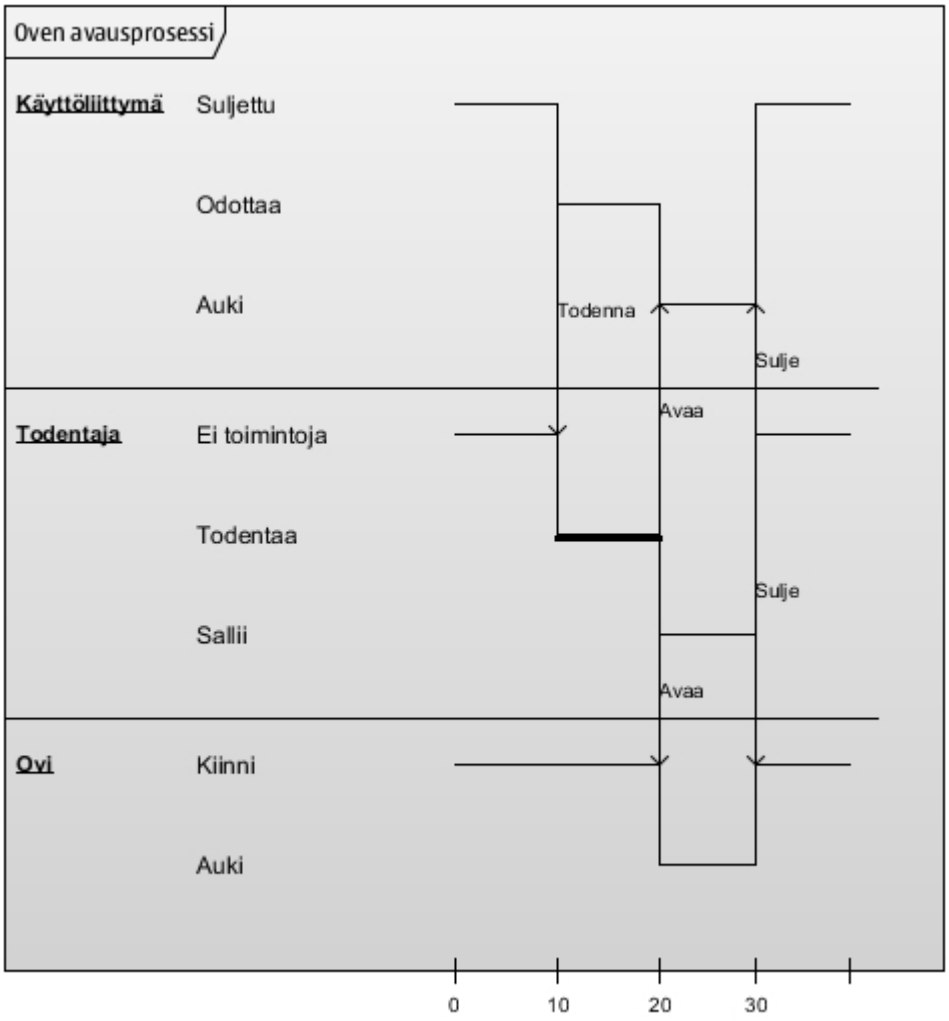
Tilakaavio kuvaa järjestelmän toiminnon tilasiirtymistä. Kaavioon osallistuu toiminnon alku- ja loppupiste, toimintoon liittyviä tiloja ja tilasiirtymisiä. Kuva 3.10 esittää esimerkin tilakaaviosta.



Kuva 3.10 Tilakaavio.

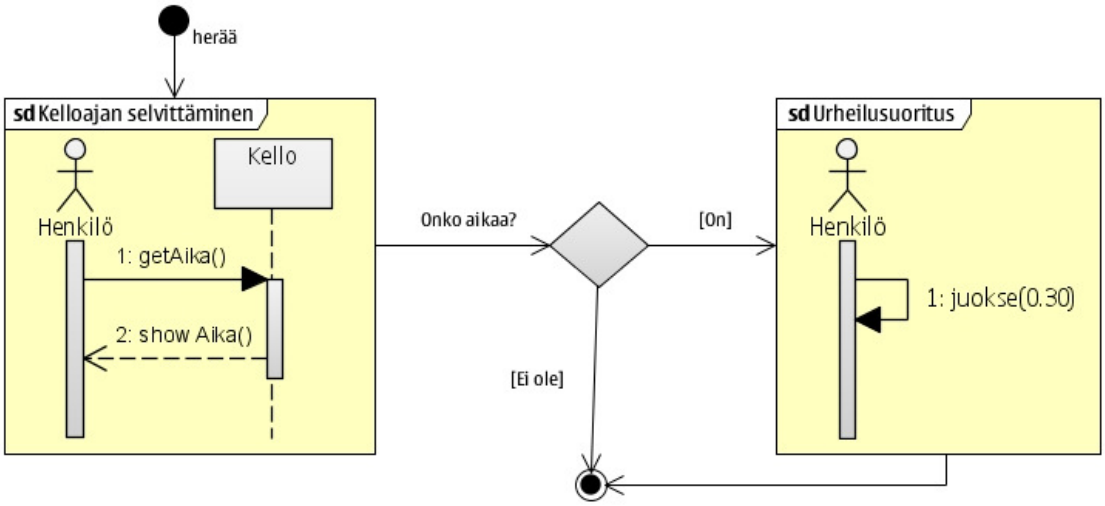
3.2.3 Vuorovaikutuskaaviot

Ajoituskaavio kuvaa järjestelmän komponenttien käyttäytymistä tietyltä aikaväliltä. Kaavioon osallistuvat muun muassa järjestelmän komponentit ja niiden funktiot. Kuva 3.11 esittää esimerkin ajoituskaaviosta.



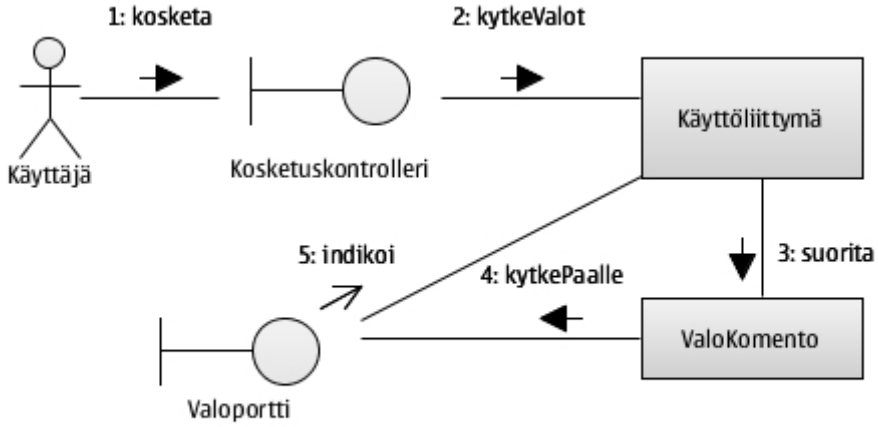
Kuva 3.11 Ajoituskaavio.

Kokoava vuorovaikutuskaavio kuvaa järjestelmän käyttäytymistä, jossa vuorovaikutuskaaviot esiintyvät solmuina (eng. *nodes*). Kaavioon osallistuvat sekvenssikaaviot, ehtokuvaukset, alku- ja loppupisteet. Kuva 3.12 esittää esimerkin kokoava vuorovaikutuskaaviosta.



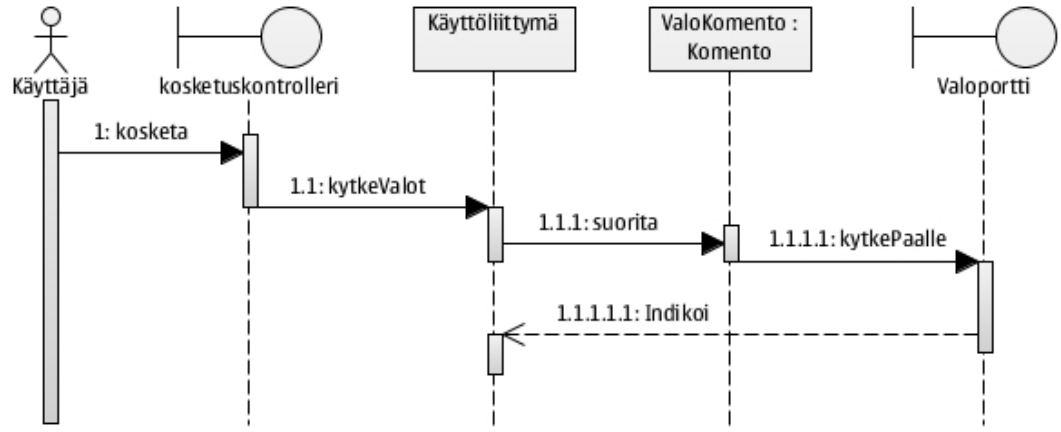
Kuva 3.12 Kokoava vuorovaikutuskaavio.

Kommunikointikaavio esittää olioiden välistä vuorovaikutusta ilman ajan mittausta. Kutsuoperaatiot on kuitenkin numeroitu järjestyksessä. Kaavioon voivat osallistua oliot, operaatiot ja toimijat. Kuva 3.13 esittää esimerkin kommunikaatiokaaviosta.



Kuva 3.13 Kommunikaatiokaavio.

Sekvenssikaavio esittää olioiden vuorovaikutusta, jolloin oliot järjestetty niiden esiintymisajan mukaan. Jokaisella oliolla on pystysuora elämänviiva (eng. *lifeline*), joka on kiinnitetty aikalaskuriin. Kaavioon osallistuvat oliot, niiden toiminnot ja elämänviivat. Kuva 3.14 vesittää esimerkin sekvenssikaaviosta.

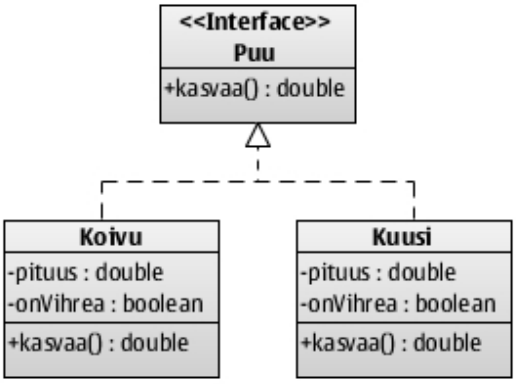


Kuva 3.14 Sekvenssikaavio.

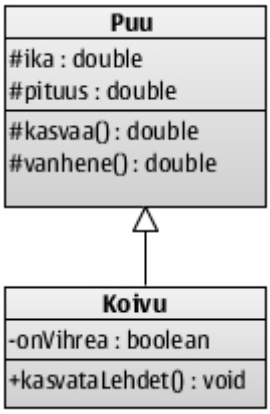
3.2.4 Relaatiot

Relaatioita eli elementtien välisiä suhteita esitetään UML-kielessä viivojen ja nuolien avulla. Seuraavassa esitetään yleisempien relaatioiden käyttöä konkreettisin esimerkein.

Toteutus merkitään katkoviivalla, jonka päässä on toteutettavissa olevaan rajapintaan osoittava tyhjä nuoli. Toteutus kertoo, että rajapinta toimii esikuvana sitä toteuttaville luokille. Kuten kuva 3.15 osoittaa, *Koivu* ja *Kuusi* noudattavat *Puu*-rajapinnan rakennetta *kasvaa*-metodin myötä.



Kuva 3.15 Toteutus.

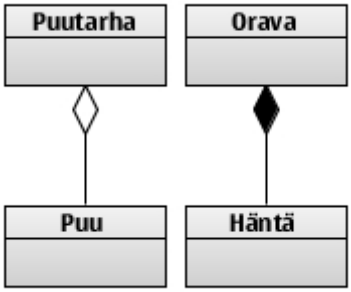


Kuva 3.16 Periytyminen.

Periytyminen merkitään viivalla, jonka päässä on isäntäluokkaan osoittava tyhjä nuoli. Relaatio tarkoittaa kuvan 3.16 tapauksessa, että *Koivu* perii kaikki *Puu*-luokan ominaisuudet, eli *Koivu* on *Puu*-tyyppinen luokka, jolla on käytössä kaikki *Puun* attribuutit ja metodit.

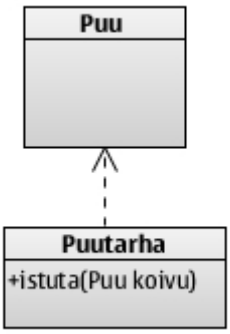
Kooste- ja muodosterelaatioita käytetään, kun halutaan ilmaista kokoelman ja sen yksittäisen osan riippuvuussuhdetta. Koosterelaatio kertoo, että rakenneosat ovat heikossa sidoksessa niiden kokoelmaan. Kuvan 3.17 esimerkki näyttää, että *Puu* tulee toimeen, vaikkei *Puutarhaa* olisikaan. Relaatio merkitään viivalla, jonka päässä on tyhjä timantti.

Muodoste on koostetta vahvempi relaatio, se kuvaa sellaisten osien suhdetta, joiden olemassaolo on mahdoton ilman kokoelmaa. Esimerkiksi *Häntä* ei tule toimeen ilman *Oravaa* (Kuva 3.17). Toisin sanoin, kun kokoelma hävitetään, hävitetään myös sen jäsenet. Relaatio merkitään viivalla, jonka päässä on täytetty timantti.

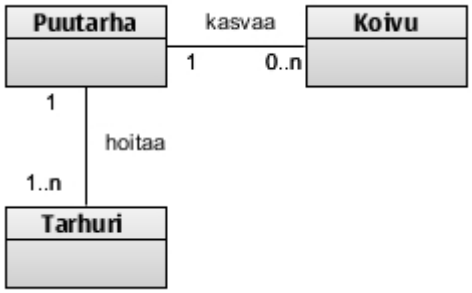


Kuva 3.17 Kooste ja muodoste.

Riippuvuus merkitään katkoviivalla, jonka päässä on kohdeluokkaan osoittava nuoli. Riippuvuus-relaatio osoittaa, että riippuva luokka käyttää kohdeluokkaa omissa toiminnoissaan. Näin olleen kuvan 3.18 *Puutarha*-luokan metodi *istuta()* riippuu *Puu*-luokasta.



Kuva 3.18. Riippuvuus.



Kuva 3.19 Assosiaatio.

Assosiaatio-relaatiolla ilmaistaan lukumääräsuhteita tietyn assosiaation ehdoilla. Kuvan 3.19 esimerkkitapaus näyttää, että *hoitaa*-assosiaation ehdolla samaa *Puutarhaa* voi hoitaa yksi tai monta *Tarhuria*, mutta jokaisen *Tarhurin* vastuulla on vain yksi *Puutarha*. Toinen tapaus kertoo, että *kasvaa*-assosiaation ehdoilla *Koivuja* voi kasvaa vain yhdessä *Puutarhassa*, mutta *Puutarhassa* saattaa kasvaa monta tai ei yhtään *Koivua*.

3.3 Ohjelmistoarkkitehtuurit

Ohjelmistoarkkitehtuuri käsittää järjestelmän rakenneosia, niiden suhteita toisiinsa ja ympäristöön sekä periaatteita, jotka ohjaavat kyseisen järjestelmän suunnittelua ja kehitystä. (IEEE 1471-standardi. Vapaa suomennos)

3.3.1 Arkkitehtuurit ohjelmistokehityksessä

Arkkitehtuurien tehtävänä ohjelmiston kehityksessä on jakaa järjestelmää komponentteihin ja tutkia niiden kommunikointia eri arkkitehtuurinäkökulmista. Koska arkkitehtuurisuunnittelua tehdään projektin varhaisessa vaiheessa, joudutaan useimmiten turvautumaan oletuksiin ja aikaisempiin kokemuksiin.

Tyypillisesti arkkitehtuurisuunnittelu tehdään projektin alkuvaiheessa, jolloin kaikki toiminnalliset ja laadulliset vaatimukset ovat määritelty. Vasta tämän jälkeen siirrytään varsinaiseen toteutukseen.

Tietojenkäsittelytieteilijä Frederick Brooks ehdotti kirjassaan The Mythical Man-Month, että ohjelmoijat kannattaa palkata vasta sen jälkeen, kun järjestelmän arkkitehtuuri on rakennettu, sillä tämä jakso saattaa venyä muutaman kuukauden pituiseksi ja ohjelmoijat jäävät työllistämättä (Brooks 1995. s.47).

Arkkitehtuurin suunnitteluun vaikuttavat halutun ohjelmistotuotteen sekä toiminnalliset että laadulliset vaatimukset, jolloin perusarkkitehtuuri rakennetaan käyttämällä ainoastaan toiminnallisia vaatimuksia, joista sitten edetään ei-toiminnallisiin vaatimuksiin.

Arkkitehtuuria voidaan rakentaa myös käyttötapausten perusteella. Tällöin on jaettava käyttötapaukset arkkitehtuurin kannalta kriittisiin ja vähemmän kriittisiin. Tämä arkkitehtuurin suunnitteluperiaate antaa käyttötapausten edustaa suoraan arkkitehtuurivaatimuksia. Vaikka perusarkkitehtuuri rakennetaan toiminnallisten vaatimusten pohjalta, sellaiset ohjelmistotuotteen aspektit kuin muunneltavuus, käytettävyys tai siirrettävyys ovat arkkitehtuurisuunnittelun keskeisimmät tekijät. (Koskimies ym. 2005.)

3.3.2 Arkkitehtuurit toteutusvälineenä

Nykyiset ohjelmistoalustat ovat kehittyneet siihen suuntaan, että kaikki perustoiminnot, kuten tiedonsiirto verkon yli tai tietokantayhteydet, on tehty valmiiksi ja ovat kehittäjän käytettävissä komponenttien muodossa. Kehittäjä keskittyy pelkästään vaatimusten toteutukseen. Lisäksi käsitys yksitäisestä sovelluksesta kadonnut toteutuksen kannalta, alettiin puhua sovelluskokonaisuuksista, jotka jakoivat yhteisiä kirjastoja ja toimivat jopa hajautetusti eri alustoilla. Arkkitehtuureista on tullut toteutusväline, jolla on ohjelmointikielen rooli. Johtuen siitä, että arkkitehtuurit toimivat korkeammalla tasolla kuin ohjelmointikieli, toteutuksessa on käytettävää tietyn alustan tarjoamia arkkitehtuureja, ohjelmointikielen sijasta.

Esimerkkinä arkkitehtuuripainotteisesta ohjelmistokehityksestä voi olla QT-graafisen käyttöliittymän toteuttaminen, joka ei itse asiassa ole C++-ohjelmointia, vaan QT:n tarjoamien käyttöliittymäkomponenttien käyttöä toteutusvälineenä (Koskimies ym. 2005).

3.3.3 Arkkitehtuurinäkymät

Järjestelmän arkkitehtuuri muodostuu järjestelmän arkkitehtuurinäkymien yhteiskuvauksesta. Näkymät mallintavat jonkun asian järjestelmän rakenteesta, ja mahdollistavat ohjelmakoodin tuottamisen automaattisesti. Kuva 3.20 havainnollistaa järjestelmän eri arkkitehtuurinäkymiä (Koskimies ym. 2005).

Kuva 3.20 Arkkitehtuurinäkymät.

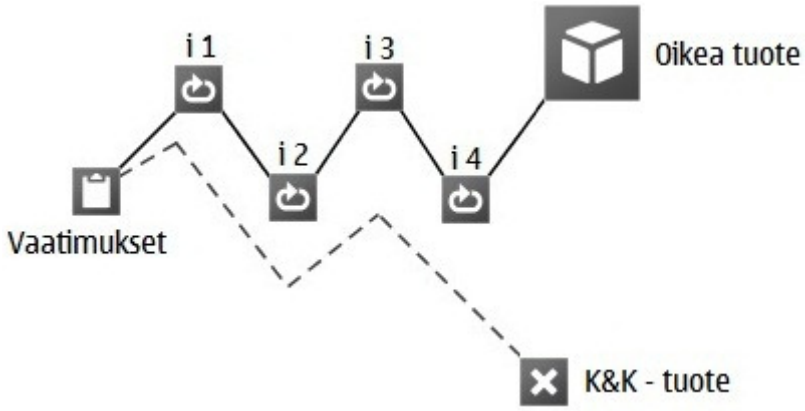
3.4 Ohjelmiston kehitysmallit ja menetelmät

Seuraavassa on esitetty ohjelmistotuotannossa käytettävät ohjelmiston kehitysmallit ja menetelmät, joiden avulla sovelluskehityksestä tulee järjestelmällistä ja tehokasta.

3.4.1 Iteratiivinen kehitys

Iteroinnilla tarkoitetaan systemaattista toiminnon tai prosessin toistamista, kunnes haluttu tulos on saavutettu. Iteraatio on iteroinnin yksittäinen askel. Ohjelmistotuotannon prosessissa iteraatioina ymmärretään projektin tarkastuskohtia, joissa verrataan tehtyä työtä asiakkaan vaatimuksiin ja varmistetaan oikea kehityssuunta aina seuraavaan iteraation asti.

Iterointiin pohjautuva kehitystapa auttaa projektiryhmää pitämään asiakaan ajan tasalla sekä varmistamaan että lopputuotteeseen liittyviä merkittäviä päätöksiä tehdään aina asiakkaan mukana ollessa. Kuvassa 3.21 on esitetty ohjelmistotuotteen iteratiivinen kehitys ja Koodaa & Korjaa (eng. *Code & Fix*)-menetelmällä valmistettu tuote.



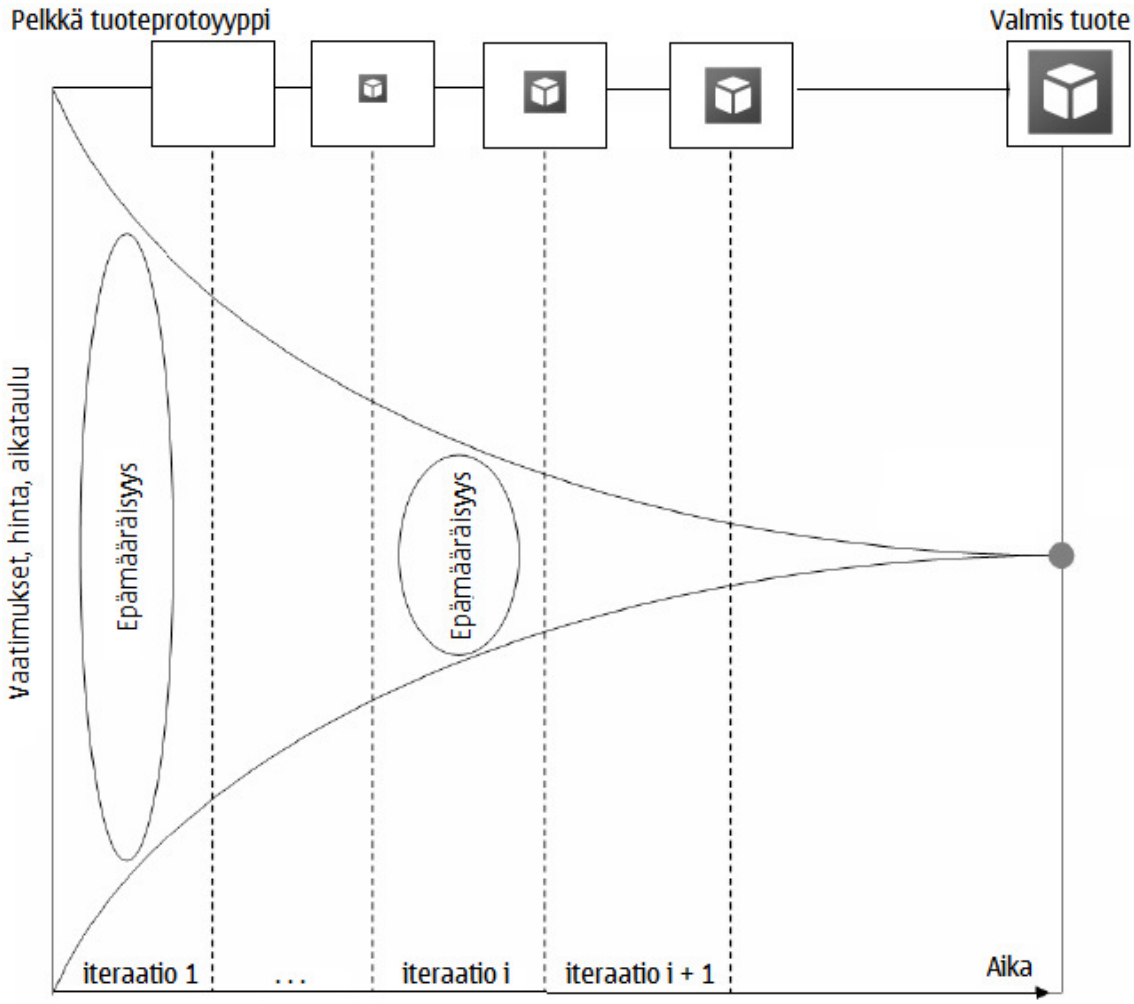
Kuva 3.21 Tuotteen iteratiivinen kehitys vs. Koodaa & Korjaa.

K&K-menetelmän mukainen kehitys voi olla alkuvaiheessa oikeansuuntaista, mutta ilman asiakastarkastuksia poikkeama oikeasta kehityspolusta kasvaa niin paljon, että lopputuote on jotain aivan muuta, mitä asiakas on vaatinut. Iteraatiota voidaan ajatella pieninä projektina suuren projektin sisällä. Yhteen iteraatioon liittyvät omat vaatimukset, resurssit, testaukset ja seuraavan iteraation asti myös aikataulu. Jokainen iteraatio toteuttaa aina samaa sykliä: suunnittelu-toteutus-tarkastus-arviointi (eng. *plan-do-check-act*).

Iteratiivinen kehitys tarjoaa mahdollisuuden tehdä äkillisiä muutoksia tuotteeseen, sillä vaatimukset muuttuvat ja kehitystavan on joustavasti reagoitava siihen. Uusi muutos saa asiakkaan tärkeysarvion ja siirtää aiemmin suunnitellut tehtävät aikataulussa eteenpäin. Tällöin asiakas on aina tietoinen, jos tuotteen valmistuspäivä siirtyy tai muutos vaatii lisäresursseja ja voi itse suunnitella muutoksia sekä olla vastuussa niistä. Iteraatiot tarjoavat lisäksi mahdollisuuden tarkistaa, että järjestelmää rakennetaan laatuvaatimuksien mukaan.

Iteratiivinen kehitystapa hallitsee hyvin myös ohjelmistotuotantoprojektin epämääräisyyttä. Monet ohjelmistoprojektit ovat alussa hyvin epämääräisiä vaatimusten, aikataulun tai hinnan kannalta.

Nämä asiat tarkentuvat iteraatioden lukumäärän kasvaessa, jolloin epämääräisyys häviää lopulta kokonaan. Kuva 3.22 havainnollistaa ohjelmistokehitysprojektin epämääräisyyden pieneneminen iteraatiomäärän kasvaessa.



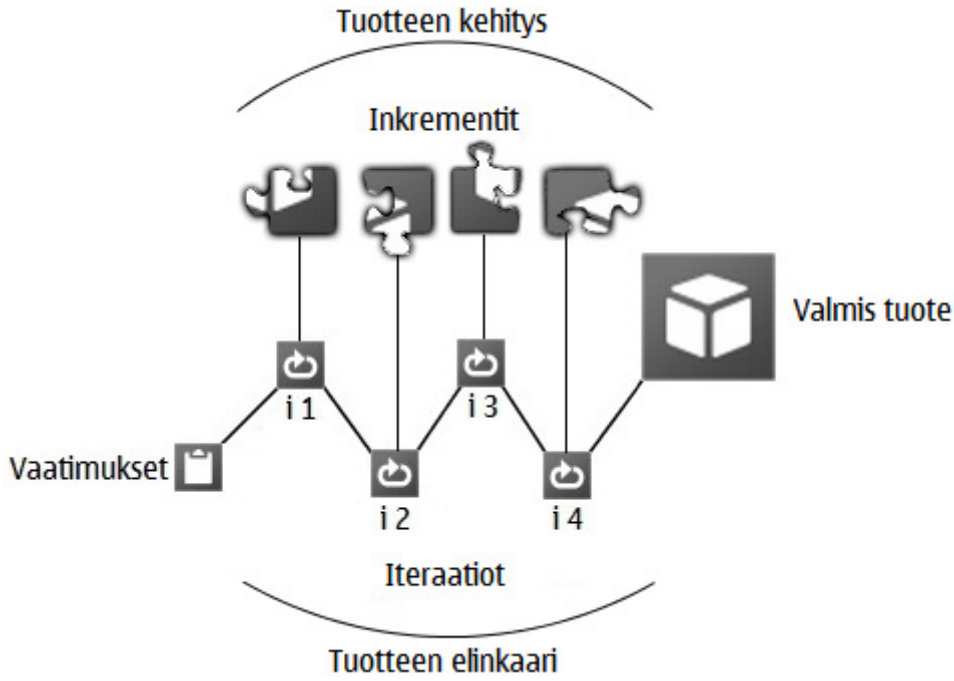
Kuva 3.22 Epämääräisyys ohjelmistokehitysprojektissa.

3.4.2 Inkrementaalinen kehitys

Inkrementaalisuudella ohjelmistotuotannossa tarkoitetaan tuotteen toimitusta osina. Inkrementaalisessa kehityksessä tuotetta suunnitellaan, toteutetaan ja testataan pala kerrallaan, kunnes tuote täyttää sille asetetut vaatimukset. Jokaisen iteraation jälkeen saadaan keskeneräinen, mutta toimiva tuote, joka julkaistaan uutena tuoteversiona. Tällöin kehitystapa sitoo projektiryhmää toimittamaan säännöllisesti uusia tuoteversioita (inkrementteja), mikä nopeuttaa lopputuotteen julkaisua.

Kun tuotteesta on olemassa yksikin toimiva versio, voidaan aloittaa testaus sekä budjetista riippuen käyttää uutta kehitysstrategiaa, jolla karsitaan turhia menoja ja säästetään aikaa. (Brooks 1995. s.268.)

Inkrementaalinen ja iteratiivinen kehitystapa kulkevat ammattilaisessa ohjelmistotuotannossa käsi kädessä, jolloin tuotetta kehitetään sykleinä (iteratiivisesti) ja toimitetaan pala kerrallaan (inkrementaalisesti). Tämä lisää projektille joustavuutta ja nopeuttaa reagointia muutoksiin. Yhdessä näistä kehitysmalleista puhutaan, että iteratiivinen kehitys ohjaa tuotteen elinkaarta ja inkrementaalinen puolestaan tuotteen kehitystä (Haikala ym. 2006). Kuva 3.23 havainnollistaa inkrementaalista ja iteratiivista kehitystapaa.



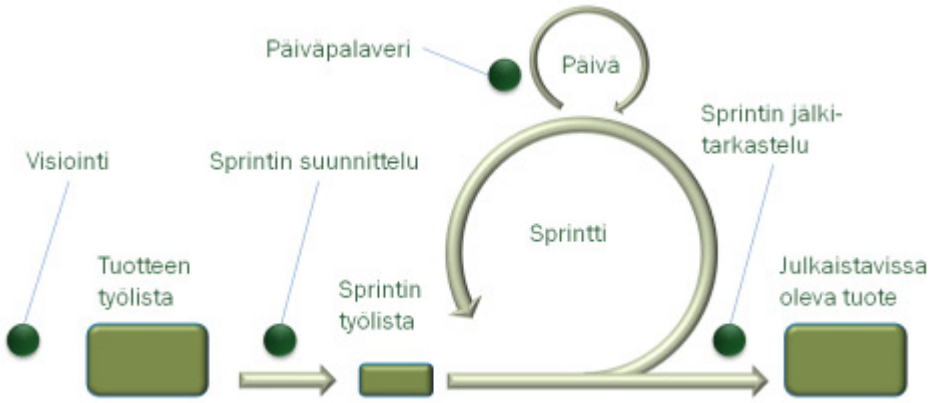
Kuva 3.23 Iteratiivinen ja inkrementaalinen kehitystapa.

3.4.3 Ketterä kehitys

Ketterä ohjelmistokehitys (eng. *Agile software development*) käsittää joukkoa ohjelmistokehitysmenetelmiä (ketteriä käytäntöjä), jotka käyttävät tiivistä ryhmätyöskentelyä sekä perustuvat iteratiiviseen ja inkrementaaliseen kehitykseen. Termi on tullut esille vuonna 2001, kun 17 ketterän kehityksen puolestapuhujaa kokoontui määrittelemään ketterien menetelmien arvoja ja periaatteita, minkä tuloksena julkaistiin Ketterä manifesti (eng. *The Agile Manifesto*).

Ketterä ohjelmistokehitys tähtää minimaalisiin riskeihin tyypillisesti viikosta, kahteen kestävillä iteraatioilla, joiden aikana nk. ketterä kehitystiimi pitää tiivistä kehityskommunikaatiota säästäen aikaa perinteisen dokumentoinnin kustannuksilla. Jokaisen iteraation jälkeen tiimi käsittelee uudelleen tehtävien työmääräarvioita, vapauttaen ylimääräisiä projektiresursseja. Ketterät kehitysmallit eivät tarjoa ”hopeaa luotia” (metafora. Brooks 1995) ohjelmistokehitykseen vaan näyttävät suuntaviivoja sen tehostamiseksi.

Eräs tunnettu ketterä kehitysmalli on Scrum (suom. *Aloitusryhmitys rugbyssä*). Mallia kuvasi ensimmäisinä *Hirotaka Takeuchi* ja *Ikujiro Nonaka* vuonna 1986. Kuva 3.24 esittää Scrum-kehitysmallin vaiheita.



Kuva 3.24 Scrum-kehitysmalli. (Poimala 2011)

Scrum aloittaa tuotekehityksen esitutkimuksella, jonka aikana selvitetään projektin tavoitteita. Projektia pilkotaan sitten yksinkertaisiin työtehtäviin, joita projektin omistaja (yrityksen asiakas tai hänen edustaja) järjestää itselleen sopivaan tärkeysjärjestykseen. Näin syntyy projektin työlista (eng. *Project backlog*). Tämän jälkeen seuraa pyrähdyksen suunnittelu (eng. *Sprint planning meeting*), jossa ketterä tiimi valitsee työlistasta yksittäisiä tehtäviä (eng. *User Story*) ja arvioi, kuinka paljon mihinkin tehtävään menee työaikaa. Näin saadaan aikaiseksi pyrähdyksen tehtävälista. Tiimi asettaa pyrähdykselle tavoitteen (eng. *Sprint goal*), joka toimii myöhemmin pyrähdyksen onnistumisen lippuna. Seuraavaksi käynnistyy noin kuukauden kestoinen pyrähdys, jonka aikana vaatimuksia ei muuteta ja kehitystiimi voi käyttää kaikkia mahdollisia keinoja käyttäjäkertomuksen (eng. *user story*) tavoitteiden saavuttamiseksi.

Oikeasuuntaista kehitystä varmistetaan kuitenkin 15 min. kestävien päiväpalaverien (eng. *Daily Scrum*) avulla, jotka toimivat kysymyskaavalla *edellisen jakson työt - seuraavan jakson työt - riskit*. Pyrähdyksen päätyttyä tiimi pitää katselmointitilaisuuden, jossa projektin omistajalle esitetään tuoteinkrementti, jonka perusteella hän päättää, vaatiiko tuote lisää pyrähdyksiä. Tiimi tarkastaa myös pyrähdyksen alussa asettamien päämäärien saavuttamista ja sitä kautta säätää seuraavan pyrähdyksen tehtävien määrää.

3.5 Suunnittelutekniikat ja periaatteet

Seuraavassa on esitetty muutamia suunnittelutekniikoita ja periaatteita, jotka antavat suuntaviivoja laadukkaan ohjelmiston tuottamiseen.

3.5.1 Antisuunnittelumallit

Antisuunnittelumallit tunnetaan huonoina ohjelmaratkaisuina, eli ”kuinka ei kannata rakentaa järjestelmiä”- ohjeina. Antisuunnittelumalleja kuten suunnittelumallejakin voi tavata kaikissa ohjelmistotyön vaiheissa. Tässä luvussa käsitellään kuitenkin vain olio-ohjelmointiin liittyviä antisuunnittelumalleja.

Antisuunnittelumallit vaikuttavat negatiivisella tavalla ohjelmiston muistikäyttöön, suorituskykyyn, vasteaikaan, turvallisuuteen sekä ylläpitoon ja laajennettavuuteen. Antisuunnittelumallien tyypillisimmät aiheuttajat ovat kehittäjän laiskuus, osaamattomuus, hätiköinti, ahneus ja jopa ylpeys. Taulukko 4.6.1-T1 esittää eniten toistuvia antisuunnittelumalleja.

Taulukko 3.1 Tyypillisimmät antisuunnittelumallit ja ratkaisut.

Antisuunn.malli	Kuvaus	Ratkaisu
God object	Luokka, jolla on liian suuri vastuu, eli liian paljon attribuutteja ja metodeja. Luokka rikkoo olioparadigman pääperiaatteen, hajota ja hallitse.	Vastuun jakaminen muille luokille. Toistuvat metodit siirrettävä abstraktiluokkiin tai rajapintoihin.

Yo-yo problem	Luokkien periytymiskartta on liian syvä. Käytetään luokkia joita peritään ja joista peritään.	Luokan täytyy toteuttaa yhden vastuun periaate. (eng. single responsibility principle). Harkittava koosterelaation käyttöä periyttämisen sijasta.
Poltergeist	Tyypillisesti luokka, jonka muodostaja herättää toista luokkaa ja häviää. Käyttää tarpeettomasti muistiresursseja.	Haamuluokan toiminnot luovutetaan muulle arkkitehtuurin kannalta sopivalle luokalle. Poistetaan haamuluokka.
Magic pushbutton	Alussa piirretään käyttöliittymä ja vasta sitten aletaan toteuttaa ohjelmalogiikka valmiiden ”eventtien” avulla esim, WindowsForm - sovellukset.	Käyttöliittymälle on tehtävä logiikkaluokat erikseen, se antaa mahdollisuuden muuttaa käyttöliittymää aiheuttamatta muutoksia ohjelmalogiikkaan.

3.5.2 Suunnittelumallit

Olio-ohjelmoinnin suunnittelumallit on kehitetty ratkaisemaan antisuunnittelumalleja tai tarjoamaan korkeamman asteen kehitysperiaatteita, jotta antisuunnittelumalleja ei syntyisi ja ohjelmistotyö olisi läpinäkyvää, jatkuvaa ja tehokasta.

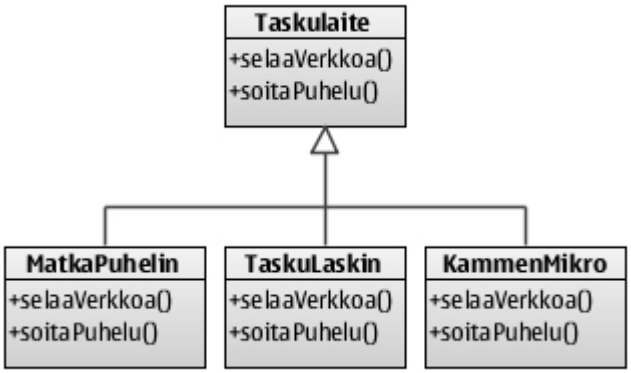
Suunnittelumalleja esitti ensimmäisen kerran itävaltalaissyntyinen arkkitehti Christopher Alexander vuonna 1977. Myöhemmin suunnittelumalleja on sovellettu ohjelmistotyöhön ja lopulta neljän hengen ryhmä, *Gang of four,* julkaisi vuonna 1995 kirjan *Design Patterns: Elements of Reusable Object-Oriented Software*, jossa käsitellään suunnittelumalleja oivana ohjelmointikäytäntönä tunnettujen antisuunnittelumallien välttämiseksi. Kirjassaan tekijät jakavat suunnittelumalleja kolmeen ryhmään käyttötarkoituksen mukaisesti:

1. *Luontimallit* (eng. *Creational patterns*) pyrkivät abstrahoimaan tavallisimpia ominaisuuksia abstraktiluokkiin tai rajapintoihin, jolloin saadaan aikaan helposti muunneltavia ja laajennettavia sovelluksia.

2. *Rakennemallit* (eng. *Structural pattenrs*) esittävät erilaisia keinoja, joilla olioille annetaan dynaaminen rakenne, minkä ansiosta niiden toiminnallisuutta voidaan laajentaa tai muuttaa ajonaikaisesti.
3. *Käyttäytymismallit* (eng. *Behavioral patterns*) tarjoavat työkaluja, joilla ohjataan ja seurataan olioiden vuorovaikutusta ajonaikaisesti.
   (Gamma ym. 1995).

Suunnittelumallit tarjoavat suuntaviivoja ongelmaratkaisuun, mutta eivät itse ratkaise ongelmaa. Sen sijaan mallit noudattavat olio-ohjelmoinnin tärkeitä periaatteita, joiden ansiosta ohjelmistosta tulee selkeä, helposti ylläpidettävä ja laajennettava. Seuraavaksi tarkastellaan nämä periaatteet konkreettisin esimerkein.

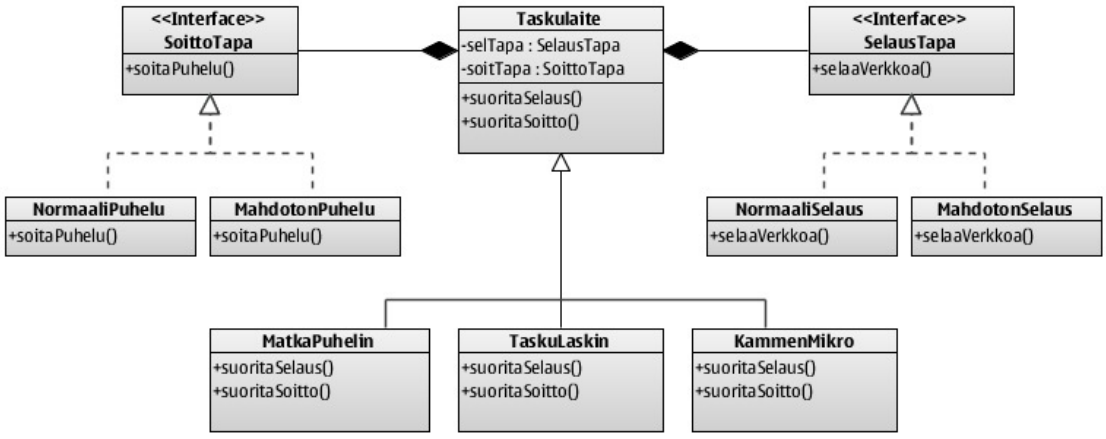
Oletetaan, että taskulaitekokoelmasta pidetään laiterekisteriä, joka erottaa laitteita kahden ominaisuuden perusteella: laitteella pääsee internetiin ja sillä voi soittaa puheluja. Tällä hetkellä rekisterissä on kolme taskulaitetta: taskulaskin, matkapuhelin ja kämmenmikro. Kämmenmikrolla ja taskulaskimella ei voi soittaa puheluja kuten matkapuhelimella, mutta kämmenmikrolla ja matkapuhelimella pääsee internetiin, mikä ei onnistu taskulaskimella. Rekisteriä voisi mallintaa seuraavanlaisella luokkakaaviolla (Kuva 3.25).



Kuva 3.25 Taskulaiterekisterin luokkakaavio.

Konkreettiset taskulaitteet peritään *Taskulaite*-kantaluokasta, joka sisältää tehtävän kannalta kaksi tärkeää metodia *selaaVerkkoa()* ja *soitaPuhelu(),* jotka ovat mahdottomia taskulaskimelle ja kämmenmikrolle. Lisäksi rekisteriin voi tulla sellaiset taskulaitteet, joilla on pelkästään puheluominaisuus.

Toisaalta sopimattomien taskulaitteiden kohdalla voidaan korvata (eng. *Override*) perittyjä metodeja, mutta huonontaa järjestelmän joustavuutta, jos pari metodia joudutaan korvaamaan tuhannessa laitteessa.



Kuva 3.26 Taskulaiterekisteri - Strategy.

Kuva 3.26 esittää ongelmaan sovellettua Strategy - suunnittelumallia, joka on hyvä esimerkki seuraavista olio-ohjelmoinnin periaatteista:

Suunnittelumallit eristävät dynaamiset jäsenet staattisista, mikä antaa mahdollisuuden työskennellä muuttuvien tekijöiden kanssa, aiheuttamatta muutoksia pysyviin tekijöihin. Taskulaiterekisterin tapauksessa muuttuva tekijä on laitteiden käyttäytyminen, joka on kapseloitu rajapintojen myötä. Näin olleen uusia soitto- ja selaustapoja voidaan lisätä koskematta konkreettisiin laitteisiin.

Suunnittelumallit suosivat koostetta periytymisen sijasta. Taskulaite-luokka koostuu eli sisältää *SelausTapa* ja *SoittoTapa* -rajapintoja, mutta ei ole itse näiden rajapintojen toteuttaja. Tämä tekee sidonnasta dynaamisen, kun taas periytymisen tapauksessa pää- ja aliluokan välille syntyy vahva sidos, joka on staattinen rakenne eikä jousta muutosten sattuessa.

Suunnittelumallit kehottavat ohjelmoimaan enemmin rajapintaa kuin toteutusta. Soitto- tai selaustapoja voi olla monenlaisia, mutta jotakin yhteistä niistä aina löytyy. Yhteiset tekijät kannattaa abstrahoida rajapintoihin, jolloin niitä on helppoa vaihtaa ajonaikaisesti (Gamma ym. 1995).

3.5.3 SOLID

Yhdysvaltalainen ohjelmistokonsultti Robert Cecil Martin esitti vuonna 2000 SOLID-olio-suunnittelun periaateyhdistelmän (lyh. *Single responsibility Open-closed Liskov substitution Interface segregation Dependency inversion*), joka auttaa luomaan ajoissa helposti ylläpidettäviä ja laajennettavia ohjelmistotuotteita.

Single responsibility principle. Luokalla täytyy olla vain yksi syy muutokseen, toisin sanoin luokalla on oltava vain ja ainoastaan yksi päämäärä, jonka varten sen kaikki attribuutit ja metodit ovat olemassa. Kuvitellaan, että edellisen luvun taskulaiterekisteriä edustava käyttöliittymä sisältää laitteiden hakemisesta vastaavaa logiikkaa. Tämä tarkoittaa sitä, että rajapintaluokalla on nyt kaksi vastuuta: edustaa taskulaiterekisteriä ja toimia hakukoneena. Jos hakuehdot muuttuvat tai rekisteriin tulee rakenteellisia muutoksia, joudutaan tekemään kaksi eri korjausta rajapintaluokkaan, entä jos hakuehtoja muutetaan kerran tunnissa? SRP-periaate ehdottaa rakentaa luokkia tai ohjelmamoduuleja niin, että olisi vain ja ainoastaan yksi syy tehdä muutoksia.

Open-closed principle. Luokat ovat auki laajennuksille, mutta suljettu muutoksilta, mikä tarkoittaa sitä, että uusi ominaisuus saavutetaan lisäämällä uusia aspekteja eikä muuttamalla vanhoja. Esimerkiksi taskulaiterekisterin soittotapoja voidaan lisätä muuttamatta niiden rajapintaa.

Liskov substitution principle. Massachisettsin teknillisen korkeakoulun professori Barbara Liskov esitti vuonna 1987 korvausperiaatteen, jonka mukaan aliluokkia on voitava sijoittaa niiden kantaluokkien tilalle ilman mitään muutoksia sovelluksen toiminnallisuuteen. Periaate saavutetaan silloin, kun aliluokka ei muuta pääluokkaa, vaan laajentaa sitä. Esimerkiksi Kalenteri-pääluokasta johdetaan aliluokka Gregoriaaninen kalenteri. Länsimaissa näiden paikkoja voidaan vaihtaa vapaasti aiheuttamatta muutoksia järjestelmään.

Interface segregation - periaate suosittelee rakentamaan monta asiakaskohtaista rajapintaa yhden ”lihavan” rajapinnan sijasta.

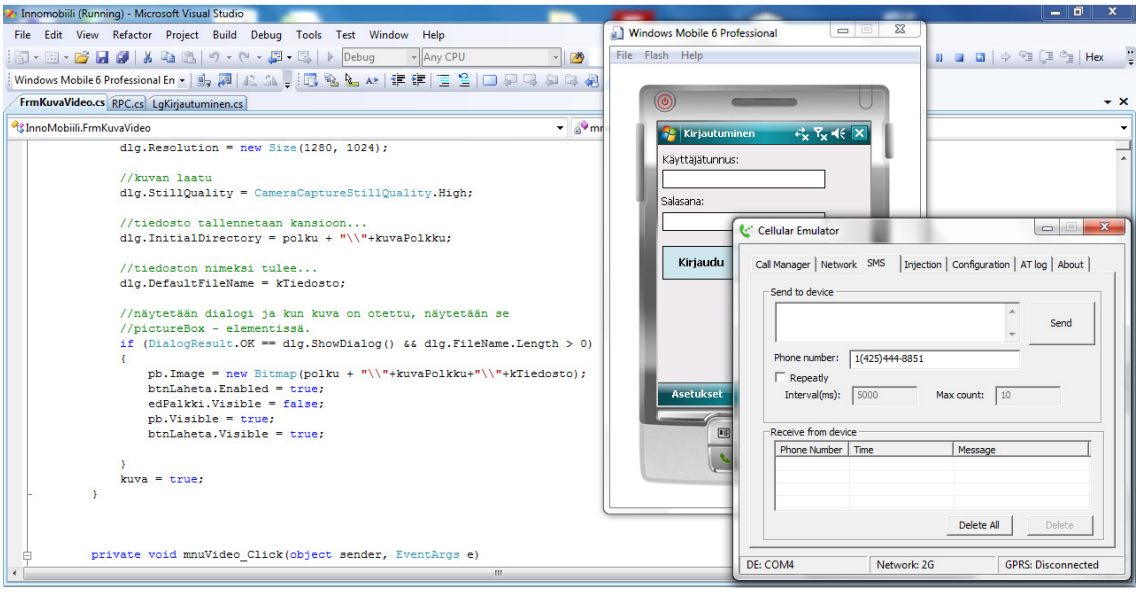
Taskulaiterekisterin tapauksessa rajapinnat on erotettu kahteen konkreettisen rajapintaan yhteisen *Tapa*-rajapinnan sijasta, sillä rajapintojen täytyy olla mahdollisimman lähellä asiakasta eikä palvelinta.

Dependency inversion. Kaikki järjestelmän sisäiset riippuvuudet perustuvat abstraktioihin, eli pääluokat eivät riipu aliluokista. Siitä huolimatta abstraktiot eivät saa riippua yksityiskohdista, eli pääluokkaan ei saa vaikuttaa aliluokan konkreettiset tekijät, vaan päinvastoin.

4 TYÖSSÄ KÄYTETYT TEKNIIKAT

4.1 MS Visual Studio 2008

MS Visual Studio 2008 käsittää ohjelmointiympäristön, jossa voidaan mallintaa, ohjelmoida ja testata erilaisia sovelluksia lukuisia ohjelmointikielijä käyttäen. Lisäksi kehitysympäristössä voidaan mallintaa käyttöliittymiä ja hallita tehokkaasti ohjelmiston resursseja. Ohjelmointikieliä ja apuohjelmia voidaan asentaa Visual Studioon erillisten liitännäisten avulla. Kuva 4.1 esittää MS Visual Studio 2008:n työnäkymää.



Kuva 4.1 MS Visual Studio 2008.

Windows Mobile Professional DTK on Microsoftin tuottama työkalupakki mobiilikehittäjälle, jonka avulla MS Visual Studiolla on mahdollista ohjelmoida, emuloida ja sijoittaa laitteeseen Windows Mobile -sovelluksia. DTK asentaa Visual mobiilisovelluksien kehityksessä.

4.2 C#

C#-ohjelmointikieli on C-kielen johdannainen, vahvasti tyypitetty hybridikieli, joka on kehitetty .NET-alustasovellusten ohjelmointikieleksi. Erilaisten tukimekanismien ansiosta C# helpottaa ohjelmointityötä, jolloin keskittyminen siirtyy olennaisen sovelluslogiikan toteutukseen. C# on syntaksiltaan yhtä selkeä kuin Java ja tehokas kuin C++. (Microsoft 2008). Kirjastojen sijasta C#-kieli käyttää viitteitä nimiavaruuksiin, jotka sisältävät tarvittavia luokkia. C#-sovellusten perusnimiavaruus on *System*, yksikään C#-ohjelmaa ei käänny ilman sitä. Kuten monissa muissa kielissä, C#-ajettavat ohjelmat alkavat *Main*-päämetodista, joka edustaa ohjelman suorittamisen alkupistettä (eng. *Entry point*). Kuva 4.2 esittää C#-kielen syntaksin *Singleton*-mallin avulla.

```
//Käytettävät nimiavaruudet
using System;

namespace TestiAvaruus //Oma nimiavaruus
{
    /* Ainokainen - luokka */
    class Ainokainen
    {
        private static Ainokainen aOlio; //Ainokaisen olio

        private Ainokainen() //Ainokaisen suojattu muodostaja
        {
            //Luontiviesti
            Console.WriteLine("Ainokainen luotu...");
        }
        //Palauttaa Ainokaisen olion, mikäli ei ole olemassa
        public static Ainokainen haeOlio()
        {
            if(aOlio == null)   //Onko Ainokaisen olio alustettu?
            {
                aOlio = new Ainokainen();   //Alustetaan Ainokaisen olio
            }
            else
            {
                //Virheviesti
                Console.WriteLine("Ei voida alustaa ainokaista toisen kerran");
            }

            return aOlio; //palautetaan
        }
    }
    /* Pääluokka */
    class Program
    {
        //Entry-point - metodi
        static void Main(string[] args)
        {
           Ainokainen singleton = null; //Ainokaisen esitys

            //Yritetään alustaa
            Console.WriteLine("1. yritys");
            singleton = Ainokainen.haeOlio();

            Console.WriteLine("\n2. yritys");
            singleton = Ainokainen.haeOlio();

            //odotus
            Console.ReadLine();
        }
    }
}
```

Kuva 4.2 C#-kielen syntaksi.

4.3 Windows Mobile -alusta

Windows Mobile -käyttöjärjestelmää kehittää ja ylläpitää Microsoft. Käyttöjärjestelmän ensimmäinen versio on ilmestynyt 20 vuotta sitten ja se pohjautui edelliseen Windows CE 5.2 -käyttöjärjestelmään. Windows Mobile -

käyttöjärjestelmän ideana on toimia pientehoisilla taskulaitteilla ja tukea mahdollisimman paljon tavanomaisen PC:n Windows-käyttöjärjestelmän ominaisuuksia. Mobiililaitteet ovat kehittyneet merkittävästi viimeisen kymmenen vuoden aikana ja sisältävät monipuolisia ominaisuuksia, joita mobiilikäyttöjärjestelmien on tuettava tehokkaasti. Kuvassa 4.3 on esitetty esimerkkilaite, joka on varustettu Windows Mobile 6.5 -käyttöjärjestelmällä.

Kuva 4.3 HTC Touch HD2 -älypuhelin. (HTC)

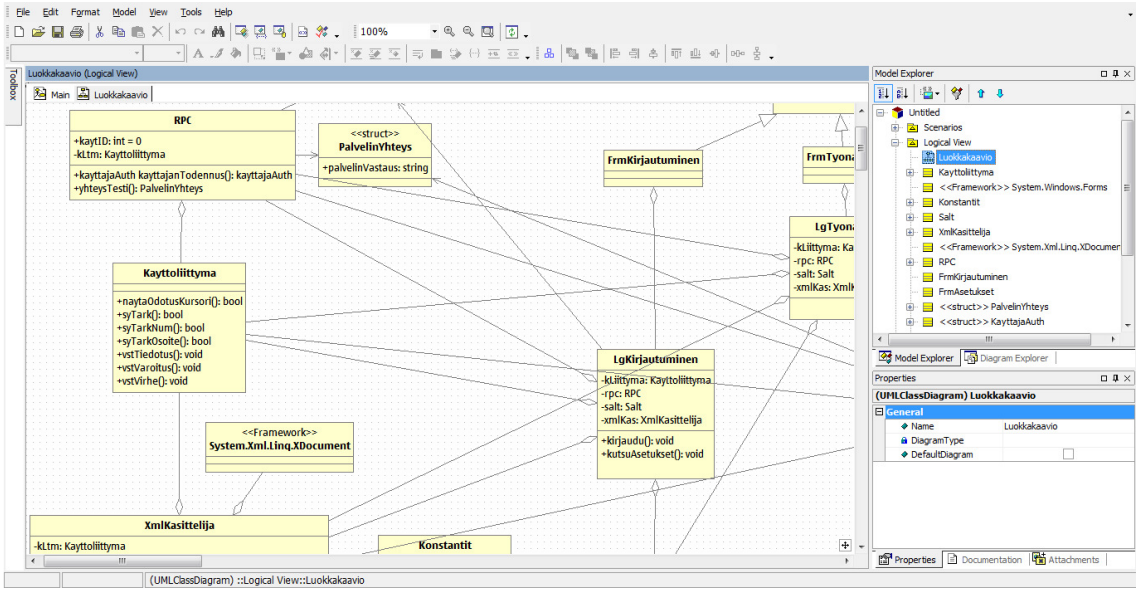
Kyseinen puhelinmalli on varustettu suurella valikoimalla monipuolisia ominaisuuksia, joista huomattavimmat ovat kapasitiivinen kosketusnäyttö, 5:n megapikselin kamera, GPS, 3G, Wi-Fi, Bluetooth sekä erikoisena ominaisuutena Maan vetovoimaan reagoiva G-sensori. HTC Touch HD2 tarjoaa kehittäjälle monipuolisen kehitysalustan, jolla voidaan testata monipuolisia ohjelmaominaisuuksia yllämainittuja laiteominaisuuksia hyödyntäen.

4.4 .Net Compact Framework

.Net Compact Framework on PC-tietokoneilla toimivan .Net Framework -kehyksen versio, joka on suunniteltu toimimaan Windows CE -alustaan perustuvissa pienteholaitteissa. Kehys sisältää .Net Framework -kehyksen valikoitujen kirjastojen lisäksi mobiilisuutta tukevia kirjastoja. .Net Compact Framework on sopiva kehys rakentaa yksinkertaisia sovelluksia, joiden käyttö ei vie paljon laiteresursseja. Tämän hetken uusin .Net Compact Famework on versiota 3.5.

4.5 StarUML

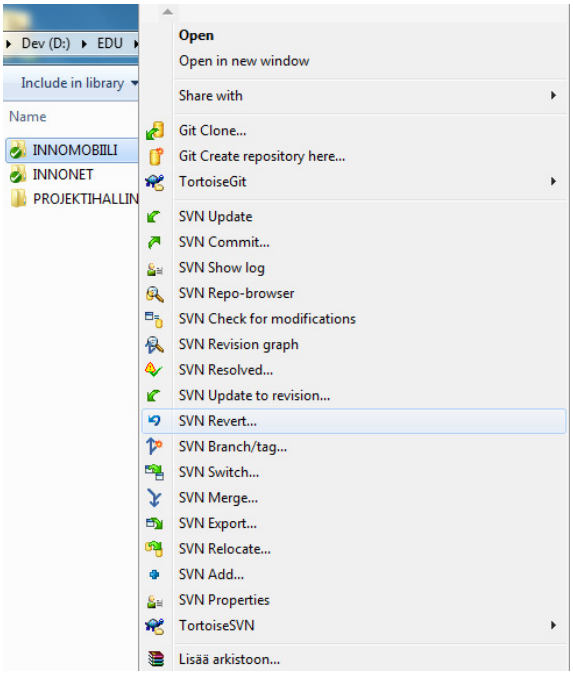
StarUML on Windows-alustalle kehitetty ilmainen UML-mallinnustyökalu, joka tukee pääasiassa kaikki UML 2.0 -standardin kaavioita. Ohjelma oli alun perin kehitetty *Delphi*-ohjelmointikielellä, mutta projekti pysähtyi pitkäksi aikaa. Nykypäivinä projekti näyttää elonmerkkejä, ja ohjelmointikielenä tällä kertaa toimii Java. StarUML-kehitysprojektin tavoitteena on korvata raskaita kaupallisia mallinnustyökaluja kuten *Rational Rose*. StarUML ei ole vielä täydellinen alusta, mutta sitä parannetaan jatkuvasti vapaaehtoisten kehittäjien voimin. (StarUML. 2005) Tämän työn kirjoitushetken viimeisin versio on stabiili 5.0, josta puuttuu vielä UML 2.0-standardin olio-, pakkaus-, ajoitus- ja kokoava vuorovaikutuskaavio. Kuva 4.4 esittää StarUML version 5.0 käyttöliittymää.



Kuva 4.4 StarUML ver. 5.0 -mallinnustyökalu.

4.6 Tortoise SVN

Subversion-versiohallintajärjestelmän avulla pidetään projektitiedostojen muutoskirjaa. Palvelimeen asennetaan versioarkistot (eng. *repository*), joita päivitetään aina kun ohjelmakoodia on muutettu merkittävästi. Versiohallintapalvelimelle tarvitaan myös asiakasohjelma, jonka avulla päivitetään versioarkistoja, seurataan versiovälistä muutosta ja ladataan versioarkistot työympäristöön. *TortoiseSVN* on eräs Subversion-asiakasohjelma, jonka käyttöliittymä on integroitu Windows Shell -kontekstivalikkoon (Kuva 4.5).



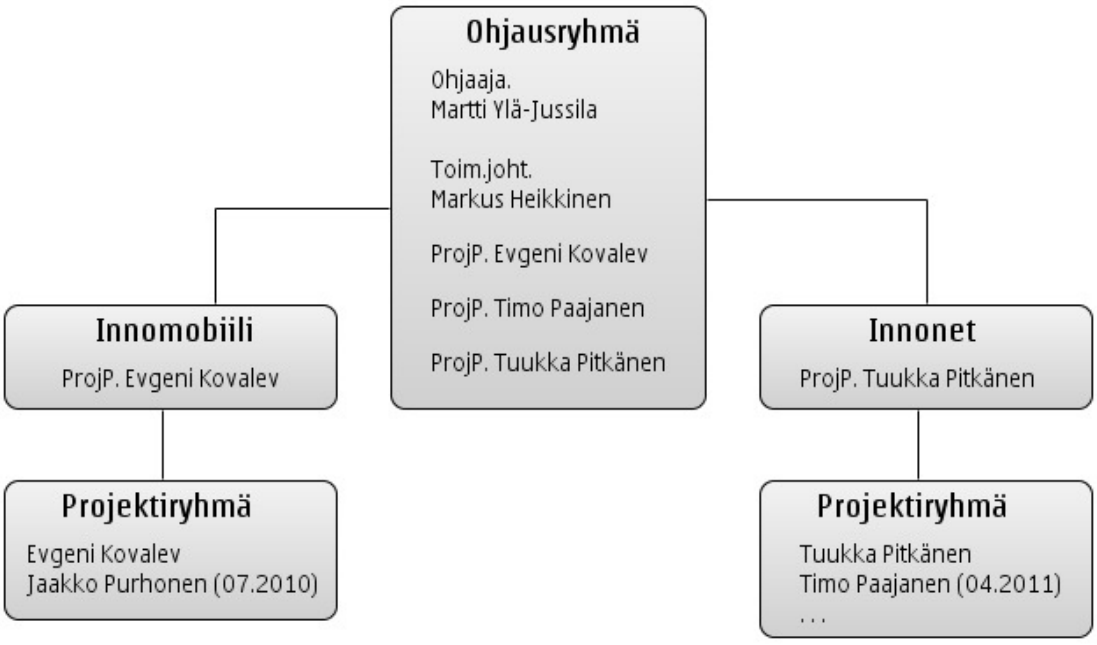
Kuva 4.5 TortoiseSVN Windows Shell -kontekstivalikossa.

TortoiseSVN on ilmainen, nopea ja helppokäyttöinen ratkaisu versiohallinnan kommunikointiin.

5 PROJEKTIN KULKU

5.1 Projektin lähtökohdat ja organisointi

Innomobiili-kehityshanke on käynnistynyt vuonna 2004 Pohjois-Karjalan ammattikorkeakoulussa, jolloin järjestelmästä tehtiin ensimmäiset koeversiot. 27. marraskuuta 2009 projekti siirtyi Saimaan ammattikorkeakouluun Jaakko Purhosen opinnäytetyöksi. Purhonen on laatinut Innomobiilista toiminnallisen määrittelydokumentin ja toiminut projektipäällikkönä 31. heinäkuuta 2010 saakka. Minun osaltani projekti on alkanut 11. toukokuuta 2010, jolloin olen toiminut projektissa suunnittelijana. Työt alkoivat tutustumisesta Purhosen määrittelydokumenttiin ja työympäristön pystyttämisestä. Yhdessä Innonet-projektiryhmän kanssa olemme käyneet ACP-videoneuvottelukursilla, minkä jälkeen 1. elokuuta 2010 Jaakko on jättänyt projektin minun vastuulle. Tästä lähtien olen ollut Innomobiili-projektin projektipäällikkönä ja ainoana työntekijänä. Projektin sidosryhmään kuuluivat henkilöt järjestäytyneet kuvan 5.1 mukaisesti.



Kuva 5.1 Innomobiili-projektin sidosryhmä.

Projektin asiakkaana on toiminut Innotek Oy:n toimitusjohtaja Markus Heikkinen. Projektissa kehitettävän tuotteen käyttäjät ovat yrityksen Energo-asentajat. Projektin ohjaavana opettajana oli lehtori Martti Ylä-Jussila. Projektin tukiryhmään kuului kaksi suunnittelijaa, Timo Paajanen ja Jaakko Purhonen. Timo Paajanen ylläpiti verkkopalvelinta, jolla sijaitsi Innonet ja Subversion-versioarkisto. Jaakko Purhonen on määritellyt Innomobiili-järjestelmän sekä johdattanut minut projektiin.

5.2 Projektiseuranta ja raportointi

Projektin aikana pidettiin ohjaajapalavereja kerran viikossa. Ohjaajapalavereissa ohjaajalle selvitettiin, mitkä työt oli tehty edellisellä jaksolla, suunniteltiin seuraavan jakson työt sekä pohdittiin projektiriskejä. Asiakaspalavereja pidettiin noin kerran kuukaudessa, jolloin annettiin selvitys tehdyistä töistä sekä varmistettiin määrittelyn päivitykset. Palaverit pidettiin verkossa Adobe Connect-verkkoviestintäohjelmalla, koska asiakas ei päässyt aina paikanpäälle.

Projektista on pidetty työaikaseurantaa, johon kirjattiin suoritettujen töiden tuntimäärät, minkä tavoitteena oli todeta alkuperäisen työmääräarvion onnistumisen.

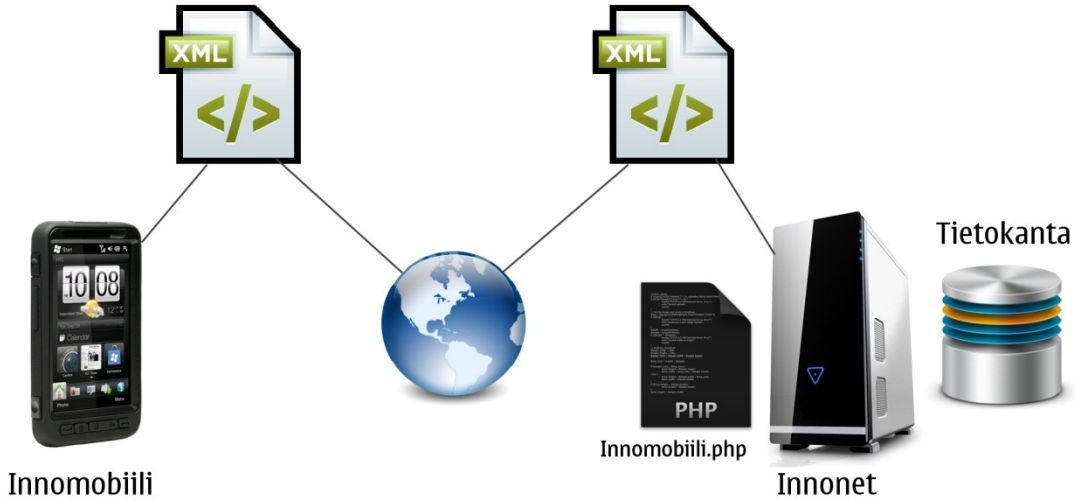
5.3 Suunnittelu ja toteutus

Alussa tarkoitukseni oli kehittää tätä projektia SCRUM-kehitysmallin mukaisesti, koska olen lukenut kirjallisuutta ja haluaisin kokeilla asioiden toimivuutta käytännössä. Valitettavasti projektiryhmään ei ole tullut ketään muuta kehittäjää, joten SCRUM jäi vähitellen pois suunnitelmistani.

Projektin aikana oli tarkoitus soveltaa opintojaksoilla opittuja asioita ja kehittää projektia mahdollisimman laadukkaasti. Toteutuksessa olen soveltanut suunnittelumalleja eri paikoissa sekä tehnyt komponenteista laajennettavia ja uudelleenkäytettäviä.

6 INNOMOBIILI-TIEDONKERUUJÄRJESTELMÄ

Innomobiili-tiedonkeruujärjestelmä on osa Innonet-toiminnanohjausjärjestelmää, joka käyttää tiedon hakemiseen ja lähettämiseen internetyhteyttä. Tiedonsiirto tapahtuu XML-RPC-mekanismin avulla verkon yli XML-muodossa. Innonet-palvelin tarjoaa etämetodeja, joita kutsuutaan Innomobiili-järjestelmän logiikkassa. Etämetodit ovat määritelty Innomobiili.php-luokkatiedostossa, joka käyttää hyväkseen toiminnanohjausjärjestelmän tietokantaa. Kuva 6.1 esittää Innomobiilin yleisarkkitehtuurin.



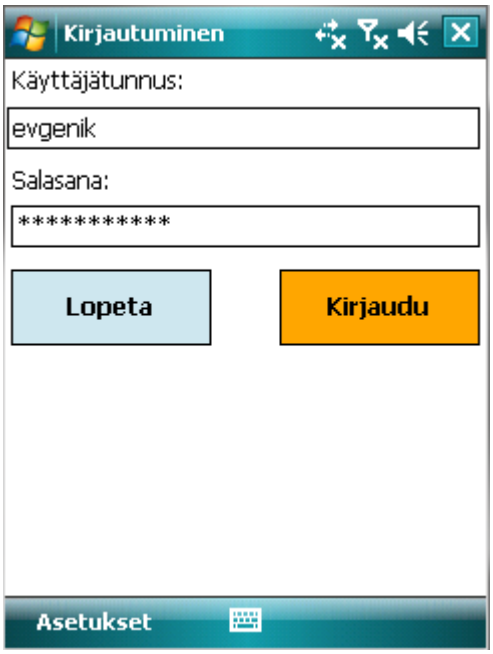
Kuva 6.1 Innomobiilin yleisarkkitehtuuri.

Innomobiilin toiminnallinen määrittely on edelleen puutteellinen, koska projekti on tällä hetkellä vielä alkukehityksen vaiheessa eikä kaikkia asioita tämän opinnäytetyön puitteessa ole ehditty määritellä tai toteuttaa. Seuraavassa esitetään tiedonkeruujärjestelmän nykyhetken version tärkeimmät ominaisuudet.

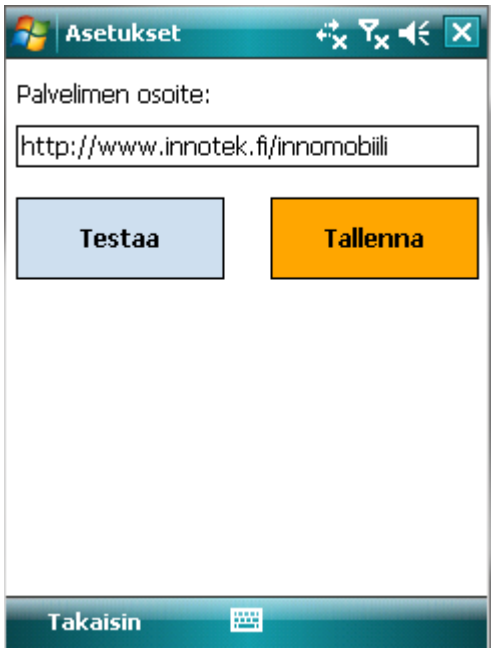
6.1 Kirjautuminen ja asetukset

Kun työntekijä käynnistää Innomobiilin, hänen täytyy kirjautua järjestelmään omilla tunnuksillaan. Työntekijä syöttää käyttäjänimensä ja salasanansa kirjautumislomakkeen avulla (Kuva 6.2). Mikäli kirjautuminen epäonnistunut virheellisen palvelinyhteyden takia, palvelimen osoitetta voidaan muuttaa valitsemalla alavalikosta kohta *Asetukset*, jolloin kuvan 6.3 mukainen asetusnäkymä tulee esiin.

Asetusnäkymän avautuessa järjestelmä lataa välittömästi palvelimen osoitteen XML-asetustiedostosta. Mikäli osoitetta ei syystä tai toisesta voitu ladata, työntekijä voi syöttää uuden palvelimen osoitteen sille varattuun tekstikenttään. Syötetyn osoitteen toimivuutta voi kokeilla painamalla *Testaa*-painiketta sekä tallentaa painamalla *Tallenna*-painiketta. Uusi osoite tallentuu XML-tiedostoon ja on järjestelmän käytettävissä aina, kun avataan uusi internetyhteys.



Kuva 6.2 Kirjautumisnäyttö



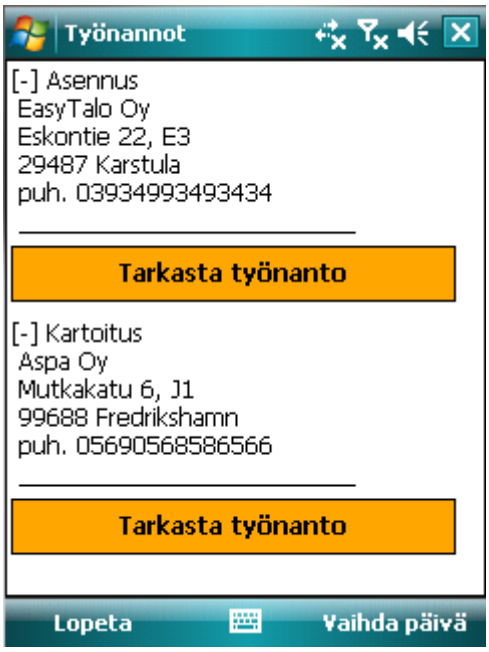
Kuva 6.3 Asetusnäyttö

6.2 Työnantojen hakeminen

Työntekijän onnistuneen kirjautumisen jälkeen Innomobiili lähettää palvelimelle pyynnön, joka hakee kyseiselle työntekijälle kirjattuja työnantoja. Edellisestä yhteyskerrasta jäljelle jääneitä työnantoja voidaan ladata laitteen XML-tiedosta. Innomobiili pyrkii hakemaan aina viimeisintä tietoa palvelimelta kirjautumisen yhteydessä. Ladatut työannot muodostavat listan, josta käy ilmi työtehtävän laatu (Energo-kartoitus tai asennus), työkohteen osoite ja puhelinnumero. Kunkin työnannon lopussa on *Tarkasta työnanto* -painike, jolla työntekijä valitsee tarkastettavaksi kyseisen työnannon (Kuva 6.4).

Kun työntekijä haluaa selata työnantoja päivän mukaan, hän valitsee alavalikosta kohdan *Vaihda päivä*, josta aukeaa kalenterinäkymä, jossa työntekijä voi klikata haluamaansa päivämäärää (Kuva 6.5).

Valittuaan päivämäärä, työntekijä painaa *Hae*-painiketta alavalikosta, jolloin järjestelmä hakee työnantolistan valitulta päivämäärältä. *Lopeta*-painiketta painamalla kirjaudutaan ulos ja lopetetaan ohjelma.



Kuva 6.4 Työnantolista.

Kuva 6.5 Työnantokalenteri.

6.3 Työn tarkastus ja valinta

Kun työnanto on valittu, järjestelmä siirtyy kuvan 6.6 mukaiseen *Työnannon tiedot* -näkymään, jossa näkyvät työnannon tärkeimmät tiedot, kuten asiakkaan yhteystiedot, työnantolaatu ja kiinteistöjen lukumäärä. Näkymän alalaidassa olevasta kiinteistöjen tarkastuspainikkeesta asentaja pääsee selaamaan kyseisen asiakkaan kiinteistötietoja *Työnannon kiinteistöt* -näkymässä (Kuva 6.7).

Työnannon kiinteistöt -näkymästä ilmenee kiinteistön tärkeimmät tiedot sekä työnannon tyypistä riippuen *Asennus*- tai *Kartoitus*-painike, josta pääsee asennus- tai kartoitustietoihin. Näkymän *Lähetä kiinteistö* -painikkeella työntekijä lähettää käsiteltyjä kiinteistöjä, joiden määrää voidaan seurata painikkeen alempana löytyvistä numeroista. Numerot osoittavat käsiteltyjen kiinteistöjen määrää suhteessa niiden kokonaismäärään (Kuva 6.7).

Kuva 6.6 Energo-kartoitus-työnanto.

Kuva 6.7 Kartoituksen kiinteistötiedot

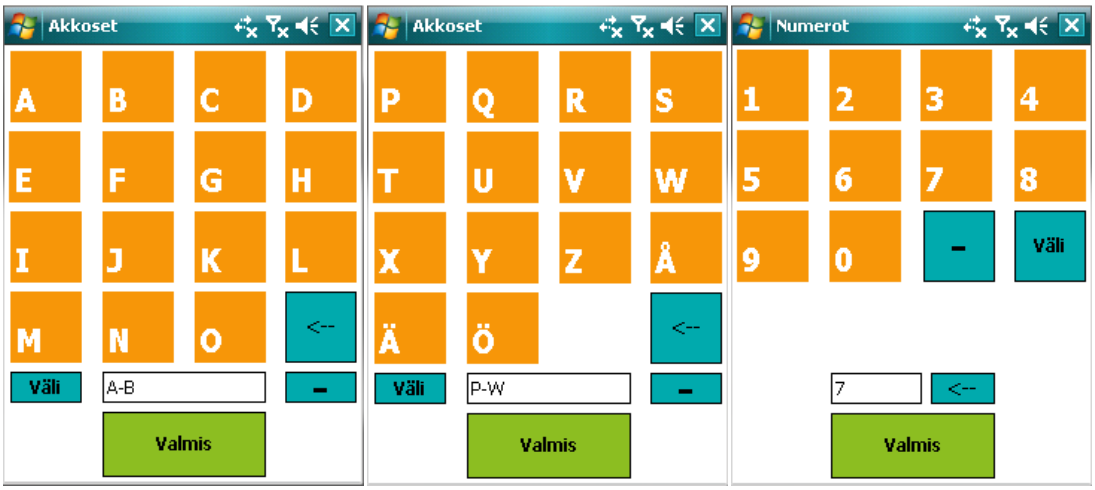
6.4 Energo-kartoitus

Energo-kartoituksen suoritus alkaa *Työannon kiinteistöt* -näkymän *Kartoitus*-painikkeesta (Kuva 6.7). Järjestelmä näyttää *Energo-kartoitukset*-näkymän (Kuva 6.8), jossa käyttäjä näkee kaikki kohdekiinteistölle tehdyt Energo-kartoitukset. Kartoitetuista kiinteistöistä on näkyvillä huoneistonumero ja tila, joka osoittaa kartoituksen valmiustilan. Jos kartoitus on suoritettu loppuun, sen tila on kuitattu. Huoneistonumero on esitetty painikkeen muodossa, mikä mahdollistaa käyttäjän tarkastaa valmista kartoitusta jälkeenpäin. Asentaja pääsee aloittamaan uuden kartoituksen valitsemalla alalaidan valikosta kohdan *Uusi*.

Kuva 6.8 Energo-kartoitukset.

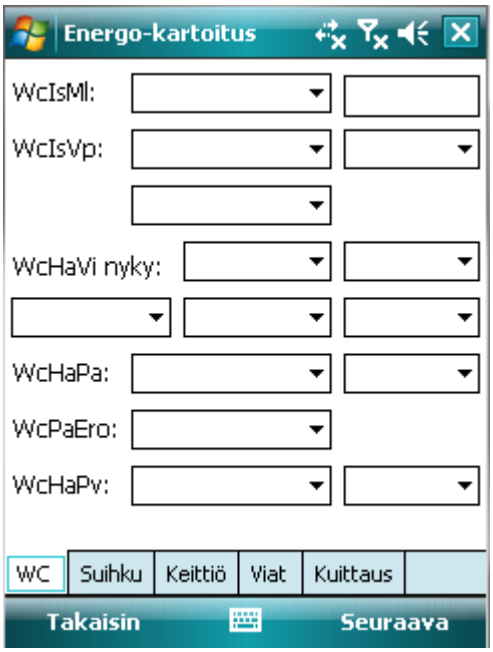
Kuva 6.9 Kartoitustila.

Valittuaan uuden kartoituksen tai huoneistonumeron käyttäjä siirtyy ensin *Kartoitustila*-näkymään (Kuva 6.9), joka sisältää kartoittavan kohteen tietoja. Näkymän avulla asentaja määrittää kartoitettavan tilan. Rakennus, rappu, kerros ja huoneisto ovat kirjain- tai numeroarvoisia tietoja. Käytettävyyden parantamiseksi näitä tietoja voidaan asettaa nuolinäppäinten avulla. Jos nuolinäppäinten käyttö ei syystä tai toisesta miellytä, tietoja voidaan asettaa aakkosten tai numeroiden syöttölomakkeiden avulla (Kuva 6.10) painamalla nuolinäppäinten välissä olevaa painiketta. Energo-kartoitukseen on mahdollista liittää kuva tai video painamalla näkymän alalaidassa olevaa *Kuvat ja videot* -painiketta.

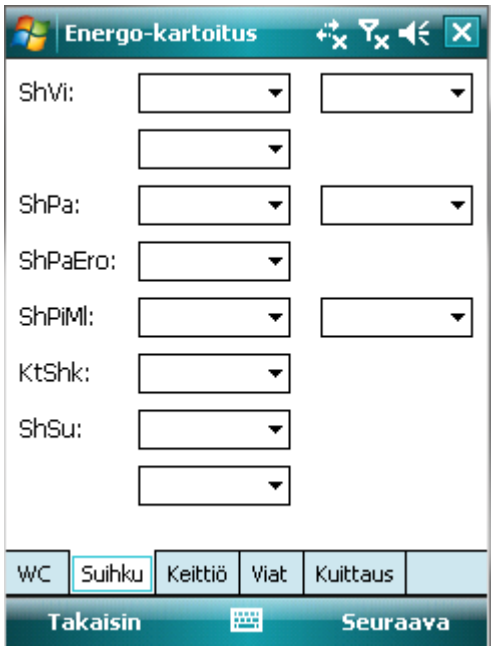


Kuva 6.10 Aakkosten ja numeroiden pikavalinta.

Kartoituksen tiedot kirjataan *Energo-kartoitus*-lomakkeeseen, jossa kartoitettavat huoneet esiintyvät välilehtielementtinä (Kuva 6.11). Kartoitettavia huoneita pääsee vaihtamaan näkymän alalaidassa olevista välilehdistä, jotka sisältävät huonekohtaisia kartoitusominaisuuksia. Näin olleen avaamalla *Suihku*-välilehden asentajan on syötettävä suihkukalusteen ominaisia kartoitustietoja (Kuva 6.12). Näkymän kartoitettavat huoneet ovat WC, suihku ja keittiö (Kuva 6.13).

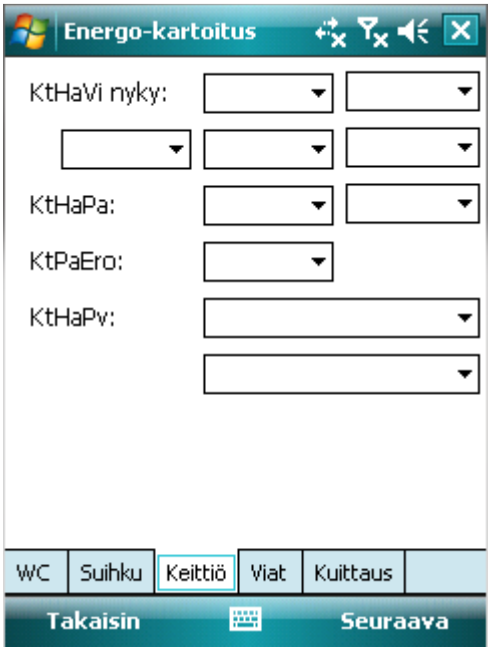


Kuva 6.11 WC:n kartoitus.

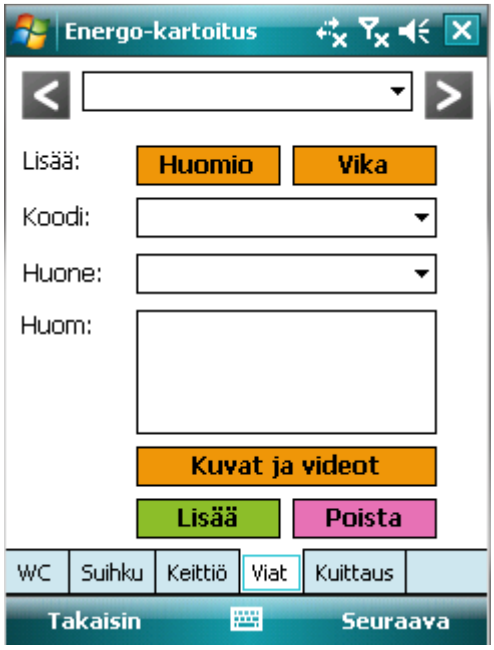


Kuva 6.12 Suihkun kartoitus.

Viat-välilehden avulla asentaja pääsee lisäämään kartoitettavaan kalustoon liittyviä vikoja tai huomioita (Kuva 6.14). Vialle voidaan antaa kooditieto, huonetunnus, vapaamuotoinen kuvaustieto sekä voidaan liittää kuva tai video.

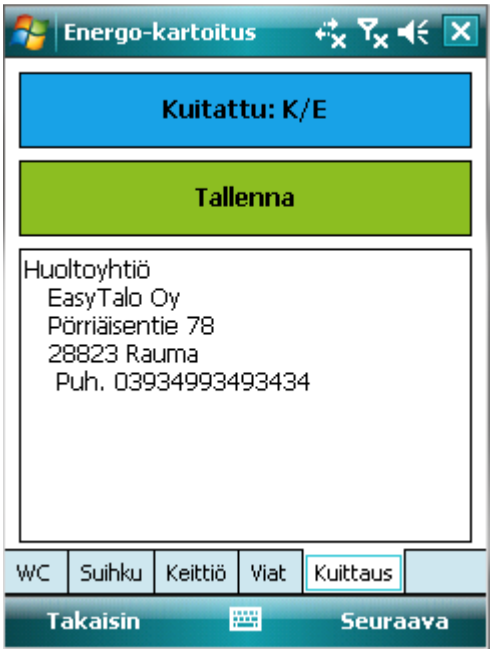


Kuva 6.13 Keittiön kartoitus.



Kuva 6.14 Kartoituksen Viat-näkymä.

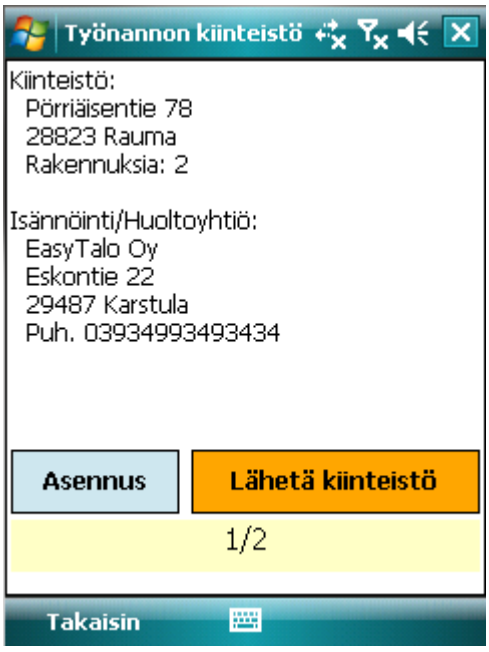
Enego-kartoituksen *Kuittaus*-välilehdessä asentaja voi kuitata kartoituksen suoritetuksi, jolloin kartoitus tallennetaan Innomobiilin muistiin ja myöhemmin lähetetään palvelimelle. Kuittauspainikkeen teksti muuttuu tapauksen mukaan *Kuitattu: kyllä* tai *Kuitattu: ei* -arvoksi. Keskeneräinen kartoitus voidaan tallentaa laitteen muistiin painamalla *Tallenna*-painiketta (Kuva 6.15).



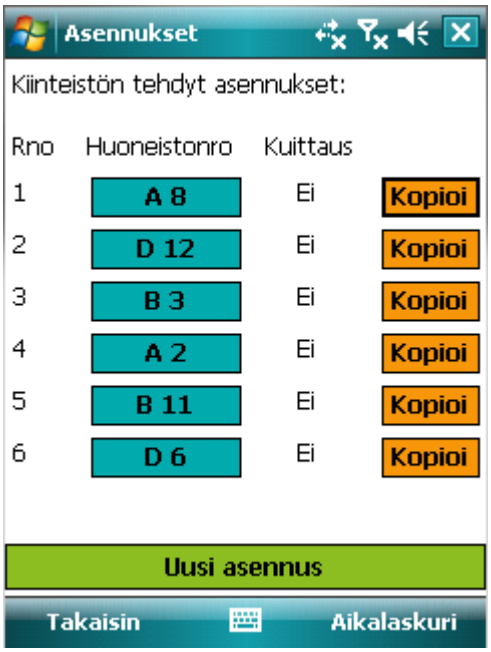
Kuva 6.15 Energo-kartoituksen kuittaus.

6.5 Asennus

Jos *Työnannot*-näkymän (Kuva 6.4) työnantotyyppi on muotoa asennus, käyttäjän eteen aukeaa *Työnannon kiinteistöt* (Kuva 6.16), josta pääsee *Asennukset*-näkymään painamalla *Asennus*-painiketta (Kuva 6.17).



Kuva 6.16 Asennustyön kiinteistöt.



Kuva 6.17 Asennukset.

Asennukset-näkymään on listattu kiinteistöön tehdyt asennukset, joista ilmenee rakennusnumero, huoneistonumero, kuittaustila sekä aikalaskuri. Uusi asennus voidaan tehdä siten, että painetaan joko *Uusi asennus* -painiketta tai painamalla *Kopioi*-painiketta, jolloin uuteen asennukseen siirtyy olemassa olevan asennuksen huoneistotiedot.

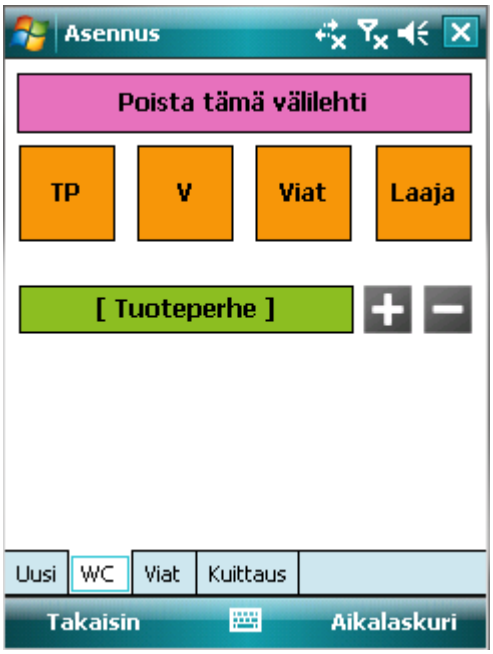
Uutta asennusta aloitettaessa käyttäjän eteen aukeaa *Asennustila*-lomake, jossa voidaan määrittää asennustilan ominaisuuksia (Kuva 6.18). Asennustilalle voidaan syöttää rakennusnumero, rappu, huoneistonumero, huoneistotyyppi sekä liittää asennusta havainnollistava kuva ja/tai video. Käytettävyyden helpottamiseksi rakennusnumeron, rapun ja huoneistonumeron voidaan määrittää valitsemalla arvon aakkos- tai numeronäkymästä (Kuva 6.10).

Kuva 6.18 Asennustila.

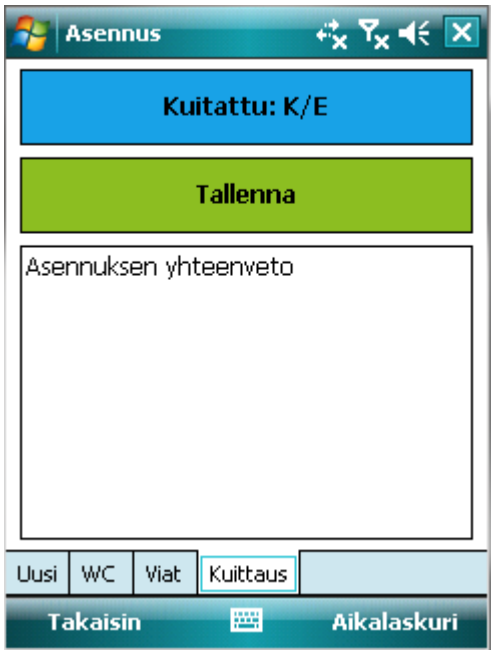
Kuva 6.19 Uusi asennushuone.

Varsinaisen asennustyön suorittamiseen pääsee painamalla *Huoneet*-painiketta, jolloin käyttäjän eteen aukeaa *Asennus*-näkymä (Kuva 6.19). Näkymän huoneet ja asennukseen liittyvät toiminnot on järjestetty välilehtinäkymään, jolloin niitä pääsee vaihtamaan valitsemalla kyseinen välilehti näkymän alalaidasta. Asennushuoneita voidaan lisätä *Uusi*-välilehden näkymässä valitsemalla ensin listasta huonetyyppi ja painamalla sitten *Luo*-painiketta. Esimerkkinä tällaisesta välilehdestä on kuvan 6.20 mukainen WC-välilehti, johon voidaan määrittää erilaisia asennustöitä ja käytettäviä varaosia.

Jokaiselle huonetyypille luodaan rakenteelta samankaltainen välilehti, jota laajennetaan tarpeenmukaisesti. Näin ollen WC-välilehdessä esiintyy esimerkiksi välilehden poistopainike, *TP*-tuoteperhepainike, *V*-varaosapainike, *Viat*-vika- tai huomiopainike.



Kuva 6.20 Asennushuoneen malli.



Kuva 6.21 Asennuksen kuittaus.

Asennukselle voidaan lisätä kartoituksen tapaan vian tai huomion liittäen siihen asiaa havainnollistavan kuvan tai videon. Kuvan 6.12 *Kuittaus*-välilehdestä asentaja pääsee kuittaamaan asennuksen suoritetuksi tai tallentamaan keskeneräisenä laitteen muistiin.

6.6 Tiedonvälitys

Innomobiilin ja Innonet-palvelimen välillä tieto kulkee *XML-RPC* -etämetodikutsutekniikan avulla. Palvelinpuolella sijaitsee php-luokkatiedosto, jossa on kuvattu *RPC*-metodit. Luokka käynnistää *RPC*-palvelimen ja tarjoaa etämetodit suoritettaviksi. *XML-RPC* on tunnetusti helppokäyttöinen ja turvallinen mekanismi, joka siirtää tietoa verkkoa pitkin *XML*-muodossa. Innomobiili kutsuu *XML-RPC*-etämetodeja *CookComputingXmlRpc*-nimiavaruuden komponentteja hyväksikäyttäen.

6.7 Reklamaatio

Jos asennetuissa tuotteissa, varaosissa tai kalusteissa ilmenee vikoja, asiakas voi tehdä viasta reklamaation. Esimerkkitapaus voi olla, että asiakas soittaa työtekijälle ja tekee suullisen reklamaation, jolloin työntekijä kirjoittaa sen ylös Innomobiilin reklamaationäkymään puhelun aikana. Asentajalla on oltava mahdollisuus kirjata reklamaatio Innomobiiliin puhelun aikana. Innomobiilissa täytyy olla mahdollisuus aloittaa reklamaation teko heti sisäänkirjautumisen jälkeen. Reklamaatiolle syötetään kohteen osoite ja päivämäärä, jonka jälkeen reklamaation tiedot tallennetaan asennuksen liitteeksi. Kun tiedot on tallennettu tietokantaan, reklamaatio on kaikkien asentajien saatavilla.

7 YHTEENVETO

Opiskelun puitteessa ennen opinnäytetyön aloittamista olin tutkinut paljon sulautettujen järjestelmien, mikro-ohjainten ja mobiililaitteiden arkkitehtuuria ja suhtauduin erityisen innokkaasti mobiilien tietojärjestelmien kehitykseen. Tämä seikka vaikutti merkittävästi myös opinnäytetyöaiheen valintaani. Olin vuosi aiemmin aloittanut tällaisen opinnäytetyön aiheen etsimistä mobiilisuutta harjoittavista yrityksistä. Onnekseni projekti löytyi, kun Martti Ylä-Jussila ehdotti, että jatkaisin Jaakko Purhosen määrittämä Innomobiili-järjestelmän suunnittelua ja toteutusta.

Työmääräarvioltaan projekti näytti kahden tai kolmen opinnäytetyön pituiselta. Asetin itselleni tavoitteen kehittää projektia hyödyntäen kaikkia opintojaksoilla opittuja tekniikoita, jolloin nopeus ei ollut olennainen tekijä, vaan laadukas lopputulos.

Aloitin Innomobiili-projektin projektipäällikkönä 11. toukokuuta 2010. Aluksi opiskelin Jaakko Purhosen tekemän määrittelydokumentin. Sen jälkeen suunnittelimme tiedonsiirtoa minulle ennen tuntematonta XML-RPC-tekniikkaa hyödyntäen.

Projektin asiakkaana toimiva Innotek Oy:n toimitusjohtaja Markus Heikkinen oli tavoitettavissa projektin aikana pääasiallisesti sähköpostin, puhelimen ja ACP:n avulla, mutta aina kun pääsi paikanpäälle, hän pystyi uhraamaan projektin hyväksi useita tunteja. Asiakkaalla tuntui olevan teknistä tietämystä siitä, kuinka tietotekniikka toimii ja vaatimukset olivat realistisia.

Projektin aikana määrittelydokumenttia on laajennettu ja täsmennetty, mutta siitä löytyy vielä monta kohtaa, jotka kaipaavat tarkennusta tai kokonaan uudelleenkirjoitusta.

Projektin riskeinä oli työvoimapula, koska työskentelin yksin eikä minulla riittänyt opintojen ohella aikaa tehdä kaikkia asioita valmiiksi seuraavan viikkopalaverin mennessä. Viikkopalavereja nimensä mukaan on pidetty viikoittain joko videoneuvottelun kautta tai pöydän ääressä.

Innomobiili toteutuksen aikana olen törmännyt siihen, että rinnakkainen Innonet -projekti ei sisältänyt vielä mitään mekanismia, joiden avulla voidaan kommunikoida Innomobiilin kanssa. Tämän vuoksi olin tilapäisesti siirtynyt kehittämään Innonettiä Timo Paajasen projektiryhmään. Tehtävänä oli perehtyä Codeigniter-sovelluskehykseen ja alkaa sitten kehittää Innomobiilin tarvitsemaa palvelinmoduulia. Näiden tehtävien ohella olen toteuttanut Innomobiilin käyttöliittymää. Siitä lähtien Innomobiili oli voimakkaasti riippuvainen Innonetistä, jonka tietokanta on elänyt jatkuvia kehitysmuutoksia.

Lopulta olen saanut Innomobiiliin osaksi toimivan käyttöliittymän sekä muutama toimiva komponentti. Lisäksi olen saanut aikaiseksi tosin keskeneräisen, mutta toimivan palvelinmoduulin, jonka kanssa Innomobiili vuorovaikuttaa XML-RPC-rajapinnan kautta. Käyttöliittymää olen rakentanut joustavaksi suunnitteluperiaatteita hyväksikäyttäen. Asettamiin tavoitteisiin olen päässyt kuitenkin vain osittain, koska uskoisin saavani enemmän näkyvää aikaan, jos en olisi projektissa yksin. Siitä huolimatta, olen tyytyväinen omaan työhön, sillä olen osallistunut kahteen projektiin yhtaikaa ja olen pyrkinyt tekemään ohjelmistotyötä mahdollisimman laadukkaasti.

Tämän ja edellisen projektin puitteissa olen oppinut näkemään ohjelmistotuotannon menetelmien positiivisia ja negatiivisia puolia. Dokumentointi ja laadunvarmistus ohjelmistotuotannossa näyttää ensisilmäyksellä ehkä projektia hidastavalta byrokratialta, mutta lopulta, tämä on sitä laatua, joka vähentää virheiden korjaustyötä loppuvaiheessa ja tuloksena saadaan tuotteen lyhyempi toimitusaika.

Innomobiili-projektin tulevaisuus näyttää kokonaisuudessaan lupaavalta. Projektin yksityiskohdat ovat aika pitkälti selvät ja seuraavan kehittäjän on suhteellisen helppoa päästä alkuun. Olen tämän yhteenvedon kirjoitusaikana etsimässä Innomobiili-projektille jatkajaa, joka olisi tarkoitus opastaa projektiin yksityiskohtaisesti. Juuri tässä vaiheessa on tärkeää saada 3–4 uutta opiskelijakehittäjää, jotka oppivat samanaikaisesti. Myöhemmin voimaan astuu Brooksin sääntö, jonka mukaan projektityövoiman lisäys myöhässä olevaan projektiin vain jarruttaa projektia.

KUVAT

TAULUKOT

LÄHTEET

Beck, K. Beedle, M. Bennekum, A. Cockburn, A. Cunningham, W. Fowler, M. Grenning, J. Highsmith, J. Hunt, A. Jeffries, R. Kern, J. Marick, B. Martin, R. Mellor, S. Schwaber, K. Sutherland, J. Thomas, D. 2001.
The Agile Manifesto. http://agilemanifesto.org (Luettu 7.2008).

Brooks, F. 1995. The Mythical Man-Month Essays on Software Engineering. Massachusetts. Addison Wesley Longman Inc.

Collins-Sussman, B. Fitzpatrick, B. Pilato, M. 2010.
Version Control with Subversion. http://svnbook.red-bean.com (Luettu 3.2011)

Durant, D & Yao, P. 2009.
Programming .NET Compact Framework 3.5 2nd ed.
Boston. Addison-Wesley.

Gamm, E. Helm, R. Johnson, R. Vlissides, J. 1995.
Design Patterns: Elements of Reusable Object-Oriented Software.
Addison-Wesley Pro Co.

Haikala,I. & Märijärvi, J. 2006. Ohjelmistotuotanto.
Helsinki. Talentum Media Oy.

HTC. Älypuhelinten valmistaja.
http://www.htc.com (Luettu 4.2011)

Innotek Oy. 2000. Esityskalvot ja lehdistötiedote.
http://www.innotek.fi/datapankki/index.html (Luettu 05.2010).

Kokkarinen, I & Ala-Mutka K. 2000. Tietorakenteet ja algoritmit.
Helsinki. Satku-Kauppakaari.

Koskimies, K. & Mikkonen T. 2005. Ohjelmistoarkkitehtuurit.
Helsinki. Talentum.

Koskimies, K. 2000. Oliokirja.
Helsinki. Satku-Kauppakaari.

Marshall, D. 2008. Programming Microsoft Visual C# 2008 The Language.
Redmond. Microsoft.

Motiva Oy. 2011. Vedenkulutus.
http://www.motiva.fi/koti_ja_asuminen/mihin_energiaa_kuluu/vedenkulutus
(Luettu 01.2011).

Oram, A & Wilson, G. 2007. Beautiful Code.
Sebastopol, CA. O'Reilly Media Inc.

Poimala, S. Heikniemi, J. Blåfield, H. 2011. Ketterät käytännöt.
http://www.ketteratkaytannot.fi (Luettu 3.2011)

Purhonen, J. 2010. Innomobiili - mobiili tiedonkeruu- ja raportointisovellus.
Theseus 2010. (Luettu 6.2010).

StarUML. 2005. About StarUML.
http://staruml.sourceforge.net/en/about.php (Luettu 3.2011).

Wikimedia Foundation, Inc. 2010. IEEE 1471.
http://en.wikipedia.org/wiki/IEEE_1471 (Luettu 11.2010).

Winter, D. 2003. XML-RPC Specification.
http://www.xmlrpc.com/spec (Luettu 10.2010).